Universidad
Carlos III de Madrid
www.uc3m.es

DESARROLLO DE LA COMUNICACIÓN USB PARA EL ROBOT TEO

GUILLERMO GIRONA GARCÍA

Profesor: Juan Miguel García Haro

GRADO EN INGENIERÍA DE TECNOLOGÍAS INDUSTRIALES

Trabajo fin de grado - Septiembre 2014

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo de Fin de Grado el día 30 de septiembre de 2014 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

SECRETARIO VOCAL PRESIDENTE

I – ÍNDICE

II - ÍNDICE DE ILUSTRACIONES

III - AGRADECIMIENTOS

A MI MADRE.

IV - RESUMEN

En este trabajo de fin de grado, elaborado dentro del proyecto TEO, del Departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid, se abordan las cuestiones relativas al sistema de comunicación USB para el robot TEO.

En primer lugar, y en relación con el estado del Arte, se analiza lo referente a los robots humanoides en general, y en particular al robot TEO, detallándose de forma sucinta su arquitectura hardware.

Se realiza un exhaustivo análisis del protocolo USB, con la finalidad de que sirva de soporte documental para posteriores estudios, así como para el desarrollo de la programación de la MBED.

A continuación se lleva a efecto un análisis sobre la MBED, recogiendo sus utilidades y características. Por otro lado, también se menciona al microprocesador LCP1768, contenido en la MBED, en donde se refieren sus características y su arquitectura en lo referente al controlador del dispositivo USB junto con los registros y comandos necesarios para su programación.

Por último, se describe la programación utilizada para la comunicación del ordenador con la MBED, detallando su sistema de funcionamiento.

Finalmente, se elaboran los resultados finales y conclusiones, indicando los trabajos futuros que se entienden procedentes.

V – ABSTRACT

In this final project work, developed within the TEO project, from the Department of Engineering of Systems and Automatic of the Carlos III University in Madrid, the matters related with the communication USB system for the TEO robot have been approached.

In the first place, and in relation with the estate of the Art, what refers to the humanoid robots in general and in particular with TEO robot, detailing its hardware architecture in a concise way.

A thorough analysis has been made of the USB protocols, in order to serve as a documentary support for later studies, as well as to the development of programming of the MBED.

Next, an analysis about the MBED has been carried out, mentioning its utilities and characteristics. On the other hand, it is mentioned the LCP1768 microprocessor, included in the MBED, with its features and architecture concerning the controller of the USB device, together with the records and commands needed for its programming.

Then, it is described the programming being used for the communication between the computer and the MBED, detailing its operating system.

To finalise, the final results and conclusions are shown, indicating the future works to be undertaken.

1 – INTRODUCCIÓN

Un robot, tal y como se define por la Real Academia de la Lengua, que consiste en "Máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a las personas", cuando es descrito como humanoide, se refiere a aquel robot que tiene una figura semejante a la humana, por lo que en principio cuenta con cabeza, torso y extremidades. En determinados casos su desplazamiento se lleva a efecto mediante ruedas, en otros se diseñan como bípedos, o bien, dependiendo de su finalidad, puede construirse únicamente la parte superior.

La investigación sobre los robots humanoides ha ido incrementándose paulatinamente en los últimos tiempos, así como su producción y desarrollo, debido por un lado a que su campo de aplicación ha aumentado, en la medida en que se le ha encontrado cada vez más utilidades, tanto para ayudar al ser humano como para reemplazarle en determinadas tareas, sirviendo de herramienta. Por otro lado, se ha perfeccionado su diseño, mejorando su técnica y ampliando sus capacidades. Tienen multitud de aplicaciones, tanto en el campo de la industria, como de la investigación o de la medicina.

Como ha expuesto el investigador en robótica humanoide, Kazuhito Yokoi, codirector del Laboratorio de Robótica Japonés-Francés, en la conferencia celebrada en la Universidad Carlos III, el desarrollo de los robots está ligado a su utilización en las tareas tediosas o peligrosas o bien las que deban realizarse en condiciones extremas, sustituyendo al ser humano, lo que, evidentemente, supone un gran avance y mejora en las condiciones de vida*:* [12] *"Los humanoides podrán sustituir al hombre en labores rutinarias y tediosas, en trabajos peligrosos, como la lucha contra el bioterrorismo o la manipulación de compuestos tóxicos, y en tareas en condiciones ambientales extremas, como el espacio".*

Por estos motivos el desarrollo de la robótica humanoide, nueva área tecnológica que impulsa la creación de mecanismos que ayuden y sustituyan a los seres humanos en determinadas tareas, alcanza gran auge

en la actualidad, invirtiéndose gran cantidad de recursos, tanto humanos como económicos, en la investigación.

El diseño de los robots humanoides se inicia en el año 1973, cuando en la Universidad de Waseda se construyó el WABOT-1, evolucionando al WABOT-2 en 1984; en Japón, con el modelo Experimental 0 de Honda del año 1986, se sentaron las bases para la construcción de ASIMO en el año 2000, que en términos generales se reconoce como uno de los robots tecnológicamente más avanzado del mundo.

[1] Dentro de los últimos desarrollos robóticos se encuentra el HRP-4, la cuarta versión de robot creado dentro del Proyecto robóticos Humanoide, patrocinado por el Ministerio de Economía, Comercio e Industria de Japón en conjunto con la Nueva Organización de Desarrollo de Energía y Tecnología Industrial, y está encabezado por Kawada Industries. Está concebido para realizar diversas actividades en el campo de la ayuda doméstica.

ATLAS, uno de los últimos robots presentados por Boston Dynamics, encuentra su utilidad en el campo de la defensa, está diseñado para reducir la cantidad de víctimas y la destrucción de bienes en conflictos militares.

THOR, creado por los investigadores de la Universidad de Pennsylvania, se desarrolló para actuar en el rescate de personas en situaciones de catástrofes naturales.

Geminoid F, robot diseñado por Hiroshi Ishiguro, de la Universidad de Osaka, con apariencia de mujer, puede reproducir expresiones faciales humanas.

NAO, desarrollado por Aldebaran Robotics, trabaja en e campo de la medicina infantil, actuando con niños autistas y colaborando con hospitales infantiles, tanto como compañero de juegos, como ayudando a superar situaciones que causan estrés.

1. 1 - OBJETIVOS

Este estudio forma parte de un proyecto más amplio, el robot TEO, del Departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid.

La finalidad de este trabajo es el desarrollo de la comunicación USB para implementarla en el robot TEO, utilizando para ello la plataforma de desarrollo MBED. Estas plataformas recogerán la información de unos encoders absolutos mediante el protocolo SPI e enviará esta información vía USB a la CPU de TEO.

Para ello, lo primero es instalar el firmware en la MBED, que está disponible en su página web. Una vez hecho esto en la MBED, ya pueden implementarse los programas necesarios para su utilización; para la elaboración de los indicados programas se emplea el compilador que ofrece MBED de forma online.

Se han utilizado las librerías que obran a disposición de los usuarios en la propia página web de la plataforma, para el empleo de la MBED como un dispositivo USB.

Una vez compilado el programa e instalado en la MBED, se utiliza el módulo pywinusb de Python, como interface para poder enviar y recibir los datos de la MBED desde el ordenador.

El objetivo final es conseguir la comunicación con todas las MBED utilizando un solo bus, esta cuestión difiere de la arquitectura convencional del protocolo USB. Se intentaron diversos métodos para conseguirlo sin resultados satisfactorios, por lo que se ofrece posibles alternativas a este problema.

Se documenta de forma exhaustiva y completa el protocolo USB y su implementación en las plataformas MBED.

2 - ESTADO DEL ARTE

2.1 – ROBOT TEO

En la Universidad Carlos III de Madrid, se está desarrollando el robot TEO, que forma parte del proyecto Rh-2, por el Departamento de Ingeniería de Sistemas y Automática. Como antecedentes se encuentran el Robot Humanoide Rh-0, que nació en el año 2004, transformándose en el año 2006 en el Rh-1, que ha obtenido logros importantísimos en el campo de la locomoción bípeda, erguida, tanto a nivel nacional como europeo.

Un robot humanoide encuentra múltiples aplicaciones, tanto en la vida cotidiana, para ayudar al ser humano en las diferentes tareas, actuando como una herramienta o para sustituirle en condiciones peligrosas o extremas, como en el campo de la medicina, experimentación o investigación. Debe ser capaz de interactuar con las personas así como con otros robots. [13] Por estos motivos se entiende que debe reunir las siguientes características físicas:

- Locomoción bípeda.
- Dos extremidades superiores, brazos, con capacidad para manipular objetos.
- Un torso, donde puedan ser colocados elementos de control.
- Una cabeza con sensores de visión y sonido.
- Autonomía energética.

Como en todos los robots, el sistema hardware se divide en 5 subsistemas funcionales: acción, información, decisión, comunicación y alimentación.

La acción se refiere al campo de la motricidad, es decir, la capacidad del robot de realizar movimientos o de transmitir sonidos.

El sistema de información se encarga de recibir los datos que el robot obtiene de los distintos sistemas sensoriales. Hay dos tipos de sensores, los propioceptivos (que se refiere a las posiciones relativas y absolutas de las articulaciones) y exceptivos (se encarga de recibir la información procedente del exterior).

El sistema de decisión, es el cerebro del robot, se encarga de generar las órdenes que el sistema de acción requiere y de procesar los datos del sistema de información, por lo que es imprescindible para coordinar estos dos sistemas.

El sistema de comunicación se encarga de transferir la información entre dos diferentes sistemas o dentro de cada sistema.

Por último, el sistema de alimentación tiene la función de suministrar potencia eléctrica a todos los dispositivos.

[14] El robot humanoide TEO dispone de 28 grados de libertad y se estima que su peso es de aproximadamente 60 kilogramos y que tendrá una velocidad de 1 kilómetro durante la caminata. Se calcula que podrá transportar objetos de 2 kilogramos de peso e incluso sentarse. Además, su altura aumenta de 1.2 metros a 1.65 metros respecto a sus predecesores, dotando al robot de un tamaño más acorde al de un humano.

Los 28 grados de libertad están distribuidos de la siguiente manera:

- Cabeza: posee dos grados de libertad, uno en el plano axial que le permite el giro de izquierda a derecha y otro en el plano frontal que le permite ver arriba y abajo.
- Tronco: también posee dos grados de libertad, y al igual que la cabeza, uno en el plano axial para girar de izquierda a derecha y otro en el plano frontal para regular su inclinación.
- Brazos: cada uno dispone de seis grados de libertad, distribuidos entre el hombro, el codo y la muñeca. El hombro tiene tres grados de libertad, en el plano axial, sagital y frontal, el codo posee un grado de libertad en el plano frontal y la muñeca dos grados de libertad en los planos frontal y axial.
- Piernas: cada pierna posee seis grados de libertad, entre la cadera, la rodilla y el tobillo. La cadera tiene tres grados de libertad, en los planos frontal, sagital y axial, la rodilla dispone de un grado de libertad en el plano frontal y por último, el tobillo posee dos grados de libertad en los planos frontal y sagital.

Con respecto a la arquitectura, hay que comentar que el sistema de decisión, está formado por dos microprocesadores principales, uno de ellos, el caminante, se encarga de enviar y recibir las órdenes de las piernas para mantener la estabilidad, y el otro, el manipulador, se encarga de asir los objetos. La necesidad de los dos microprocesadores viene dada para la distribución de la información entre las piernas y los brazos, reduciendo así la carga de información y los tiempos de consulta, procesamiento y acción.

El sistema de acción se compone de los motores y de la transmisión. Cada motor es controlado por un driver que interpreta las órdenes que el microprocesador correspondiente a las piernas o brazos le envía. También se utiliza como interfaz para la comunicación con el microprocesador a través del CAN BUS.

Con respecto al subsistema de información, cada motor lleva incorporado un sensor relativo que analiza la posición relativa al eje del motor y por otro lado se añade un enconder absoluto a cada motor para poder medir la posición absoluta (la real). Por otro lado, los sensores de fuerza-par situados en los pies y manos se comunican con la CPU a través de una tarjeta PCI. Por último, los demás sensores, como el de contacto o inercial son conectados a la CPU a través del USB, del RS232 o de entradas digitales de que disponen los microprocesadores.

El sistema de alimentación, además de la fuente, se compone de los convertidores y del cableado. Es autónomo, de poco peso y controlado por el sistema.

Con respecto al sistema de comunicación, los dos microprocesadores están conectados por el protocolo TCP/IP al exterior mediante Wifi. Por otro lado, se crea una conexión punto a punto entre los puertos LAN de los microprocesadores, a fin de que sea una conexión más eficaz, veloz y sin problemas de colisión de mensajes. Para esta comunicación se utiliza el protocolo Ethernet. Para comunicar los driver de los motores con los microprocesadores se utiliza el protocolo de comunicación CAN. Se utilizan 3 buses CAN; dos de ellos se conectan al microprocesador caminante y el otro al manipulador. Por otro lado, para enviar la información de los encoders absolutos del sistema manipulador a su

microprocesador se hace también por medio del CAN. Por último, y es en lo que está centrado este proyecto, para enviar la información de los encoders absolutos del sistema caminante a su microprocesador, los encoders envían primero la información a las MBED a través del SPI y posteriormente las MBED envían la información recibida vía USB al microprocesador.

En la siguiente imagen se muestra un esquemático de la arquitectura de TEO, indicando en la parte inferior derecha de cada componente la cantidad de los que dispone el robot.

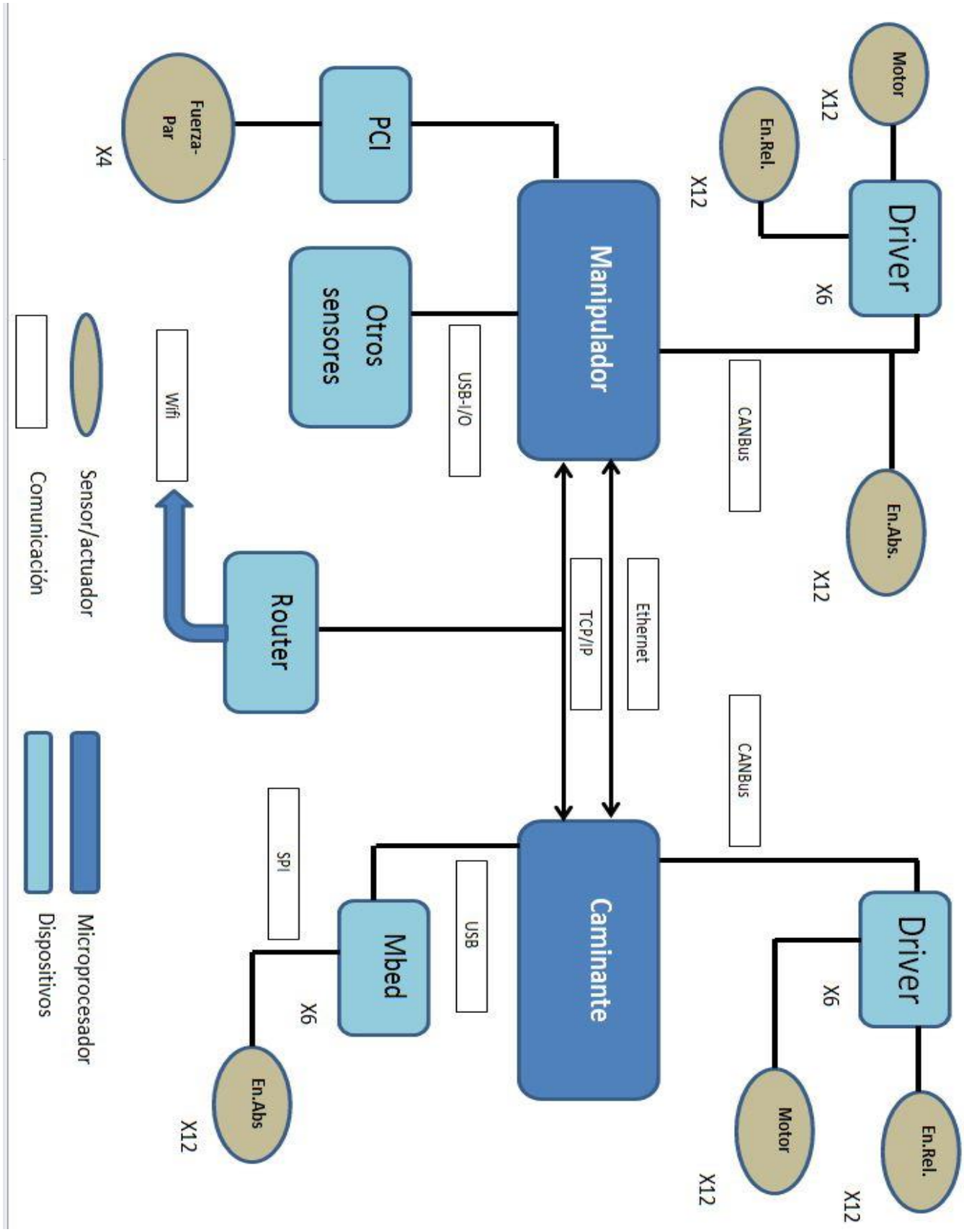


Ilustración 1: Arquitectura hardware TEO

2.2 - PROTOCOLO USB

2.2.1 – INTRODUCCIÓN

Un bus es un subsistema que transfiere datos o potencia eléctrica entre los componentes de un ordenador o entre ordenadores. Dentro de estos se encuentra el USB (Universal Serial Bus) que consiste en una norma estandarizada que determina las utilidades empleadas en un bus para conectar y comunicar ordenadores, periféricos y dispositivos electrónicos, de tal forma que todos los dispositivos USB deben tener el mismo tipo de cable y de conector, con independencia de la función que deban desempeñar.

Su desarrollo nace de la necesidad de conectar distintos dispositivos al equipo informático. La primera versión "USB 1.0" que fue creada en el año 1996, no fue bien recibida, no siendo hasta el año 1998 con la especificación "USB 1.1" que alcanzó gran reconocimiento. Desde entonces su utilización se ha incrementado exponencialmente, habiéndose implementado distintas versiones. Se puede decir que desde el año 2004 se encuentran en el mercado 6 mil millones de dispositivos, calculándose que las ventas anuales globalmente consideradas alcanzan la cantidad de dos mil millones.

El USB se utiliza para la conexión al ordenador de distintos periféricos, tales como ratón, teclado, cámaras digitales, memoria USB, escáner impresoras, tarjetas de sonido, discos duros externos, etc. Dada su gran eficacia, ha desplazado a otros sistemas de conexión que han quedado absolutamente obsoletos.

En la actualidad su aplicación se extiende a cualquier dispositivo electrónico o con componentes, tales como la radio de un automóvil, reproductor de Blu-ray Disc, algunos juguetes como el Dinosaurio Pleo. Ha sido desarrollado con variaciones para su implementación tanto a nivel industrial como militar, así como en el campo de la telefonía móvil.

La ventaja del USB frente a otros conectores radica en que permite la conexión de dispositivos al ordenador sin necesidad de reiniciarlo ni de volver a configurar el sistema, ya que se configuran automáticamente una vez que se ha conectado físicamente, siendo identificados por el ordenador,

pudiendo ser desconectados mientras que el equipo informático esté en uso. A esto se le denomina plug and play.

Por otro lado, la actividad del USB se fundamenta en el software, en concreto por el sistema operativo, por lo que son más fáciles de fabricar y más baratos.

Otra de las ventajas del USB es que tiene la capacidad de alojar en un mismo bus dispositivos con diferentes velocidades de transmisión. En la siguiente imagen se muestra las velocidades que hay, a qué velocidad transmiten, sus aplicaciones y sus características.

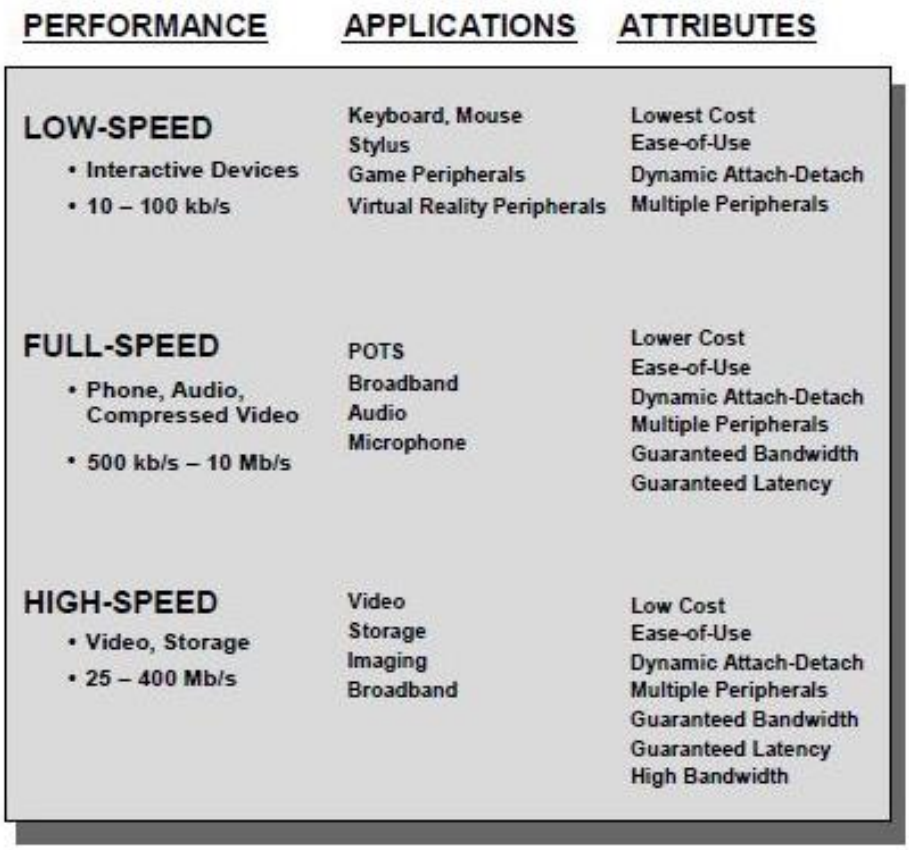


Ilustración 2: Velocidades dispositivos USB

2.2.2 – ARQUITECTURA USB

En el sistema USB se distinguen dos componentes básicos, por un lado está el Host y por otro los dispositivos.

Solo hay un host por cada sistema USB. El host está a cargo del bus e interactúa con los dispositivos USB a través del controlador host (Host controller). Las funciones del host tanto en hardware y software son:

- Detectar la conexión o desconexión de los dispositivos USB
- Enumerar y configurar los dispositivos
- Dirigir los datos entre el host y los dispositivos
- Recolectar información sobre los estados, capacidades y las actividades de los dispositivos
- Proveer de alimentación a los dispositivos.

[4] Debido a que el host es el único que tiene el control del bus, la comunicación entre los dispositivos y el host se hace mediante la denominada cola de transferencia.

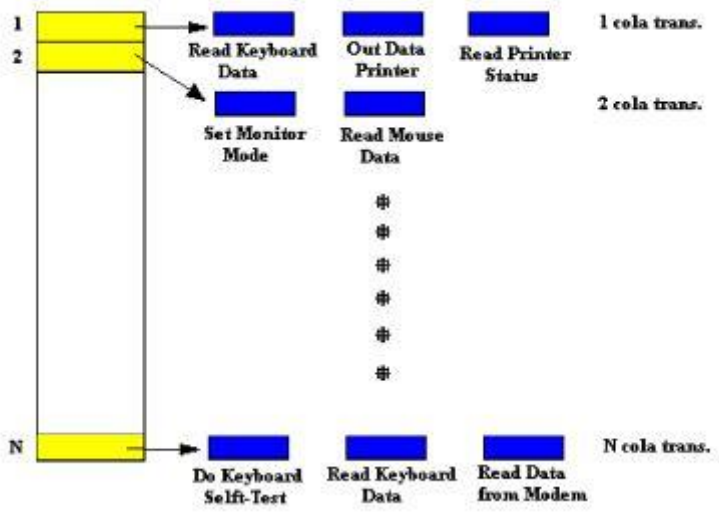


Ilustración 3: Cola de transferencia del host

Las acciones del host, tales como enviar y recibir datos de los dispositivos, se colocan y se van haciendo una a una, y son llevadas a cabo a por el método FIFO (first input first output), es decir, se van llevando a cabo por orden de entrada a dicha cola.

En relación con los dispositivos, hay que decir que cualquier dispositivo USB tiene que ser capaz de:

- Comprender el protocolo USB
- Responder a operaciones estándar como resetearse y configurarse.
- Poder dar una descripción de que tipo dispositivo es.

A todos los dispositivos se les asigna una dirección (address) cuando son conectados.

Todos los dispositivos USB tienen que soportar al menos una pipe, (representación abstracta de la comunicación entre un buffer en el host y un endpoint en el dispositivo), llamada Default Control Pipe, pero pueden soportar más.

Al menos una de las pipes está designada al endpoint (buffer de memoria, localizado en los dispositivos y se puede decir que es el final de la comunicación entre el host y el dispositivo) zero, en el que se realiza el control del USB cuando es conectado.

A esta pipe de control se le asocia la información requerida para describir el dispositivo USB.

Se distinguen dos tipos básicos de dispositivos USB:

- Funciones, que tienen una utilidad específica tales como un ratón, una memoria o una pantalla de ordenador.
- Hubs, que proporcionan conexiones adicionales para los dispositivos.

Una función tiene que ser capaz de transmitir y recibir información en el bus, se trata de un dispositivo periférico que se conecta a un puerto de un Hub; sin embargo, existen paquetes físicos que incorporan un Hub y múltiples dispositivos, por ejemplo una pantalla con altavoces, a los que se conoce como compound device, a estos el host los reconoce como un Hub con uno o más dispositivos “non-removable”. En la imagen se ve como ejemplo de compound device el monitor o el teclado (KBD), ya que tienen otros dos dispositivos conectados cada uno.

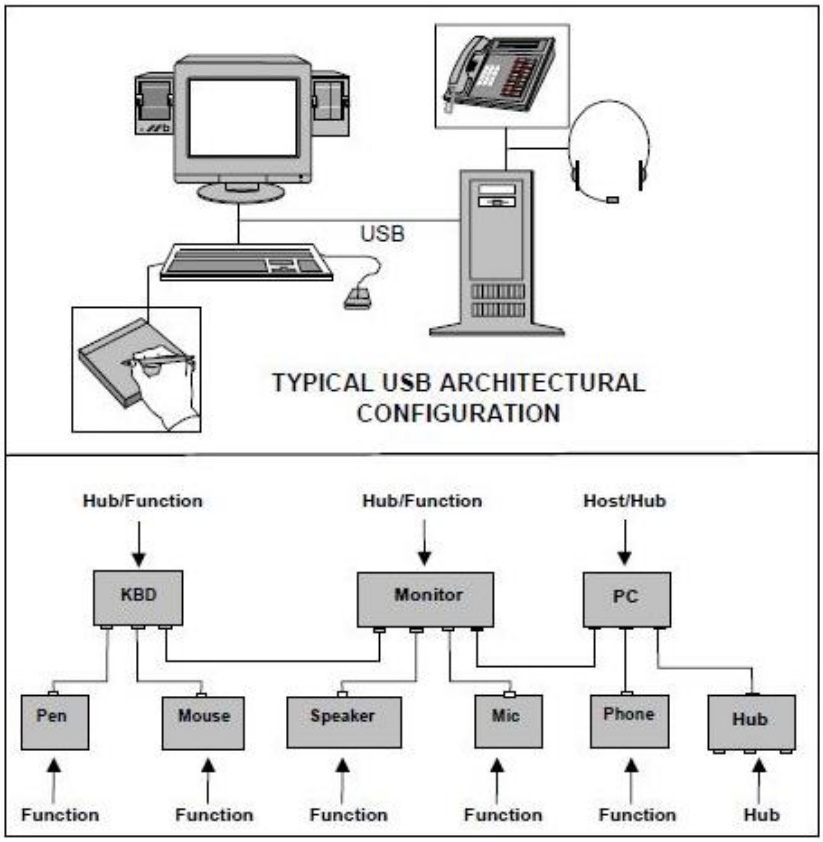


Ilustración 4: Representación de la conexión de múltiples dispositivos

Antes de que una función pueda estar operativa, tiene que ser previamente configurada por el host.

Hay tres tipos básicos de funciones:

- Human Interface, como un ratón o un teclado
- De imagen, como una impresora o una cámara
- De memoria, como un pendrive.

Por otro lado están los Hubs que son la clave para el plug-and-play (conectar y usar) del USB.

Simplifica la conectividad desde el punto de vista del usuario y proporciona robustez al sistema, a un bajo coste y con relativa simplicidad.

Básicamente un Hub posibilita la conexión de múltiples dispositivos USB a sus puertos.

Un Hub puede ser conectado a otro Hub y así sucesivamente hasta tener un máximo de 127 dispositivos.

El puerto upstream está conectado (mediante otros Hub) al host y los puertos downstream están conectados a los otros dispositivos (una función u otro Hub).

Los downstream de un HUB están capacitados para trabajar a high-, full- o low– speed. En la siguiente figura se muestra un Hub con 7 puertos downstream.

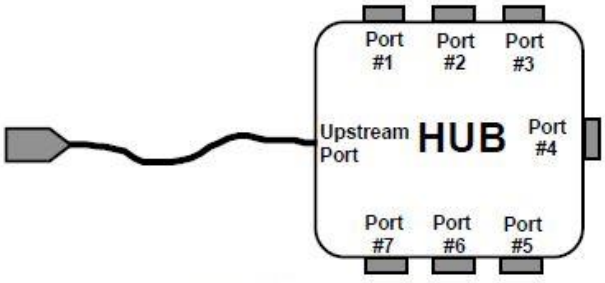


Ilustración 5: Representación HUB

Como se ve en la imagen un Hub consta de tres partes:

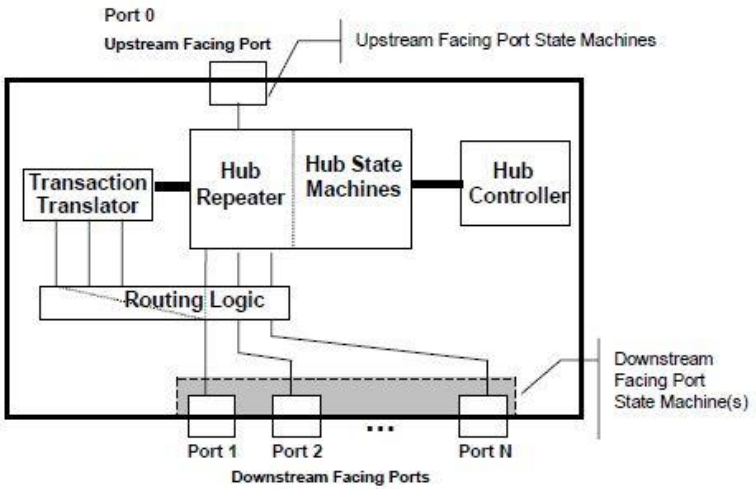


Ilustración 6: Arquitectura HUB

- El Hub controller: permite el acceso del host al Hub y proporciona el control y los estados del Hub.

- Hub repeater: maneja la comunicación entre los upstream ports y los downstream ports.
- Transaction translator: provee los mecanismos para que el Hub pueda soportar dispositivos de full/low – speed en sus downstream, ya que en su upstream es por high-speed en los Hubs posteriores a la versión 2.0.

Como se ha dicho anteriormente, el sistema USB puede trabajar a varias velocidades a la vez y eso es gracias a los Hubs y en concreto a la parte "Transacion Translator" mencionada arriba. En la siguiente imagen se muestra la configuración según las velocidades de comunicación que puede haber en un sistema USB. Aunque no aparece explícitamente en la imagen; un dispositivo FS/LS también puede ir conectado a un HS Hub.

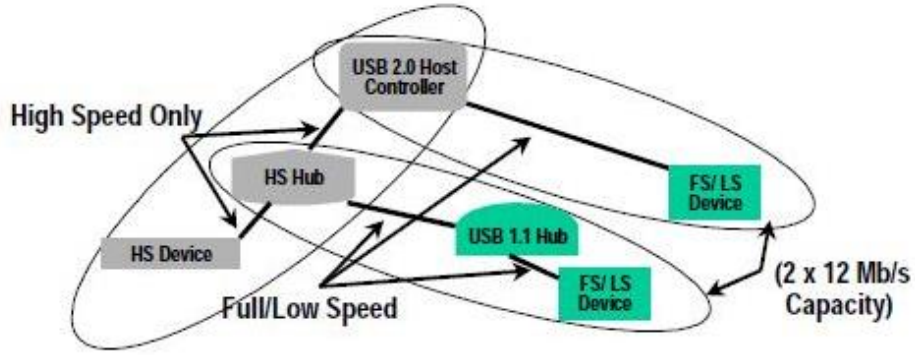


Ilustración 7: Configuración velocidades

La interconexión física entre los dispositivos y el host es una topología en estrella por niveles, donde un Hub es el centro de cada estrella. El Root Hub es el Hub que permite la conexión del host con varios dispositivos.

Cada segmento de cable es una conexión punto a punto entre el host y un Hub o entre un Hub y un dispositivo (otro Hub o una función). En cada conexión se eleva un nivel la posición.

Como máximo puede haber 7 niveles (tier). En los dispositivos compuestos (Hub + función) hay que tener en cuenta que ocupan dos

niveles, por lo que no pueden ser conectados al nivel 7, a este nivel solo pueden estar conectadas las funciones.

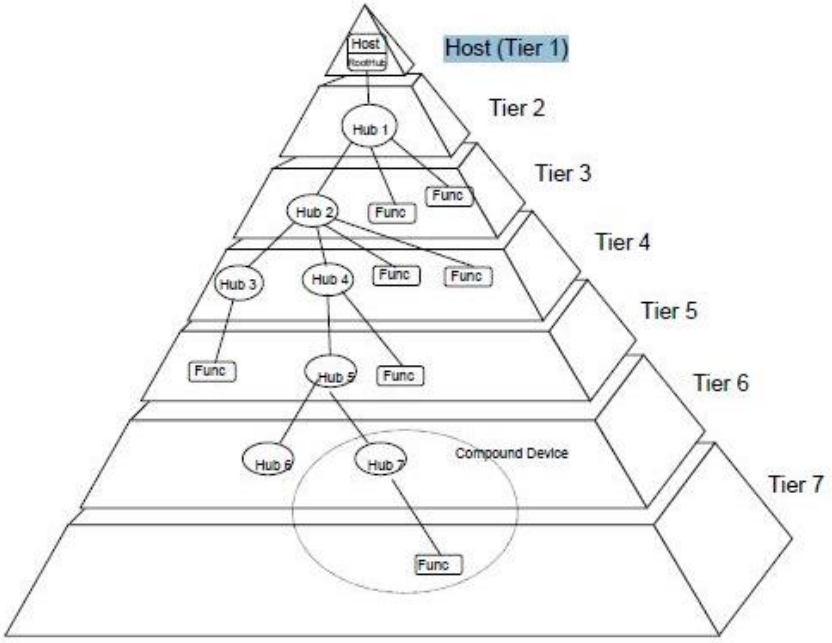


Ilustración 8: Topología del bus

2.2.3 – CAPA FÍSICA

Todos los dispositivos tienen puertos de conexión upstream con el host y, por otro lado, el host y los Hubs tienen puertos de conexión downstream para los dispositivos. Los conectores downstream y upstream no son mecánicamente intercambiables. Comúnmente hay dos tipos de conectores el Type A y el Type B (aunque hay más tipos de conectores).

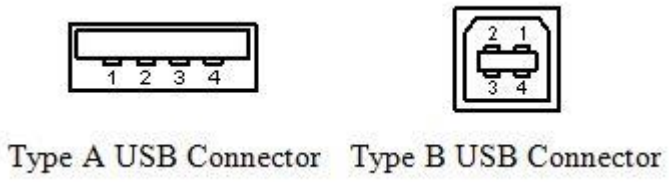


Ilustración 9: Conectores USB Type A y Type B

El tipo A es siempre para upstream, los conectores de tipo A se encuentra típicamente en los host y en los Hubs. Por otro lado, los conectores de tipo

B siempre están conectados downstream por lo que son los que se encuentran en los dispositivos.

Recientemente, en la especificación On-The-Go que permite que un dispositivo pueda funcionar como esclavo y como maestro (host y dispositivo) dependiendo de la situación, se han incluido la especificación de otros puertos tipo B, llamado conector mini.

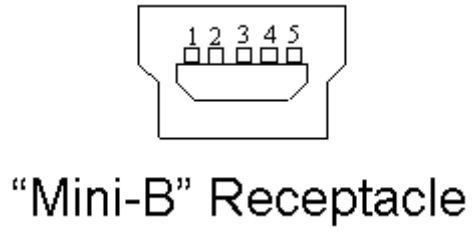


Ilustración 10: Conector USB Mini-B

Los cables USB están compuestos internamente por 4 cables. Las señales del USB se transmiten por par trenzado, cuyos hilos se denominan D+ y D- Los otros dos cables son el de tierra GND (las señales D+ y D- toman como referencia de nivel esta señal) y el VBUS que proporciona potencia a los dispositivos que lo necesiten.

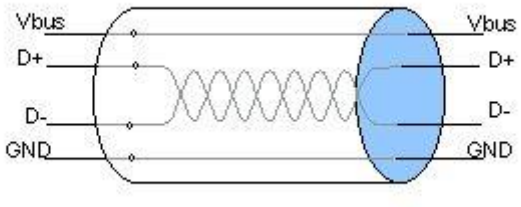


Ilustración 11: Cable USB

A diferencia de otras comunicaciones como la RS232, en el que la información tiene dos canales TX y RX, uno para enviar y otro para recibir datos, en el sistema USB la señalización es diferencial y en un solo sentido, es decir la información se envía y se recibe por los cables D+ y D- que van cambiando su polaridad.

Para poder hacer más fácil la identificación de los cables se ha definido un color para cada cable como se muestra en la siguiente figura .

Contact Number	Signal Name	Typical Cable Colour
1	VBUS	Red
2	D-	White
3	D+	Green
4	GND	Black
Shell	Shield	Drain Wire

Ilustración 12: Tabla cable USB

En el sistema USB los dispositivos pueden obtener la potencia del bus sin necesidad de conectores externos.

Al configurarse un dispositivo indica la cantidad de potencia-intensidad que van a consumir y después no puede aumentar su consumo. En función de la intensidad consumida se pueden diferenciar tres tipos de dispositivos:

- Low-power bus powered functions: Según las especificaciones este tipo de dispositivos no puede consumir más de 1 unidad de carga (100 mA)
- High-power bus powered functions: este tipo de funciones no pueden consumir más de 1 unidad de carga (100 mA) hasta que ha sido configurado; después pueden consumir 5 unidades de carga (500 mA).
- Self-powered functions: estas funciones están alimentadas mediante un conector externo, en todo caso, pueden consumir del bus hasta 1 unidad de carga.

Como hemos dicho el USB utiliza para comunicarse una transmisión diferencial. Esto es codificado usando el método NRZI (non-return-zero-inverter); en este código cuando no hay cambios en la señal se considera un "1" y un "0" es representado mediante un cambio en la señal. En la siguiente imagen se muestra abajo la señal física y arriba la señal ya codificada.

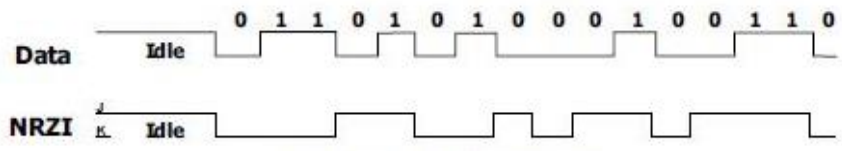


Ilustración 13: Representación codificación NRZI

También utiliza un bit stuffing para asegurar la adecuada transmisión de datos. Un bit stuffing consiste en añadir un "0" por cada seis "1" consecutivos. Si el recibidor del paquete detecta que hay 7 o más "1" seguidos rechaza el paquete ya que ha detectado un error en el bit stuffing. El recibidor del paquete tiene que estar preparado para para decodificar los datos NRZI reconociendo el bit stuffing.

En dispositivos de LS un "1" diferencial es transmitido poniendo la señal D+ por encima de 2.8 V con una resistor pulled a tierra de 15K ohm y D- por debajo de 0.3 V con una resistor pulled de 1.5K ohm a 3.3V.

Un "0" diferencial es transmitido poniendo D- por encima de 2.8 V y D+ por debajo de 0.3 V con las mismas pull down/up resistors.

En los dispositivos HG/FS cambian las resistors pulled como se explicará más adelante.

El recibidor considera un "1" diferencial si D+ es 200 mV mayor que D- y por otro lado considerara un "0" diferencial si D+ es 200 mV menor que D-

[3] Para simplificar se han definido dos estados del bus: el estado J y el estado K. Estos dos estados son opuestos; es decir si J significa un "0" diferencial, entonces K significa un "1" diferencial. Dependiendo de a qué velocidad estemos operando FS/LS o HS, los estado J y K cambian su polaridad. En FS/LS el estado J es un "0" diferencial mientras que en high speed es un "1" diferencial.

La polaridad del estado J es la misma que la del estado Idle, que es el estado del bus antes y después de que un paquete sea enviado.

También está el estado SE0 (single ended zero) que se produce cuando ambas líneas D+ y D- están por debajo de los 0.3 V. El estado SE1, que es un estado ilegal, se produce cuando ambas líneas están a alto nivel.

Cuando ningún dispositivo está conectado a los puertos downstream de un Hub o de un host las señales D+ y D- presenta el estado SE0.

La señal de conexión: La señal de conexión se produce cuando el Hub detecta el cambio de estado de SE0 a estado idle (J). El tiempo de conexión está comprendido entre 2 y 2.5 microsegundos.

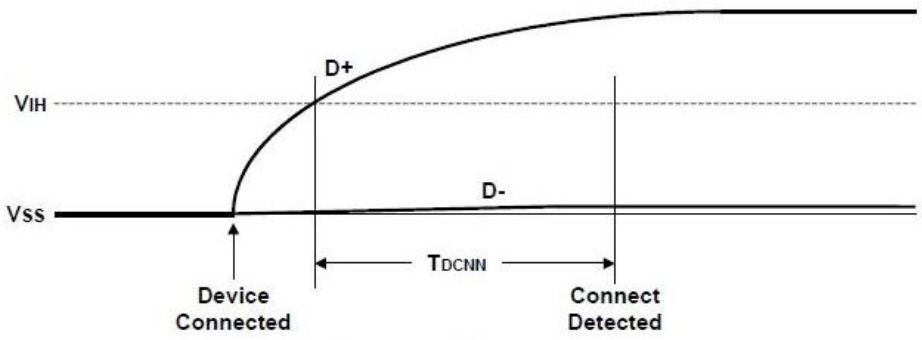


Ilustración 14: Señal de conexión

La señal de desconexión: La señal se produce cuando el Hub detecta el cambio del estado idle al estado SE0. El tiempo de desconexión tiene que ser mayor que 2.5 microsegundos.

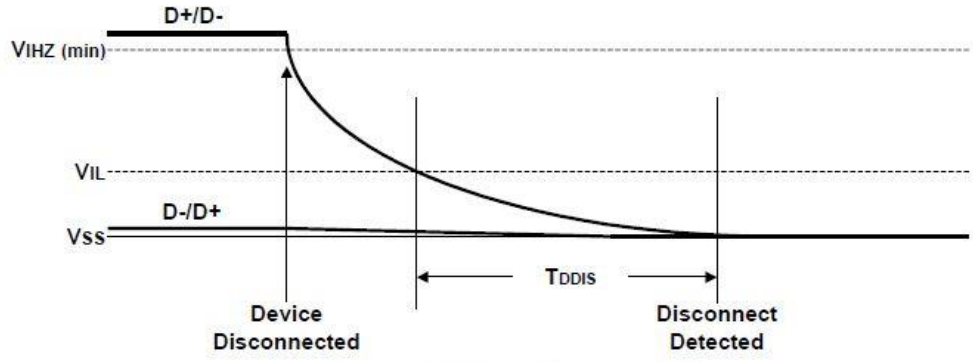


Ilustración 15: Señal de desconexión

La señal de reset: es la que se utiliza para inicializar la comunicación con un dispositivo. Como se ve en la siguiente imagen, consiste en poner el estado SE0 por lo menos durante 10 ms.

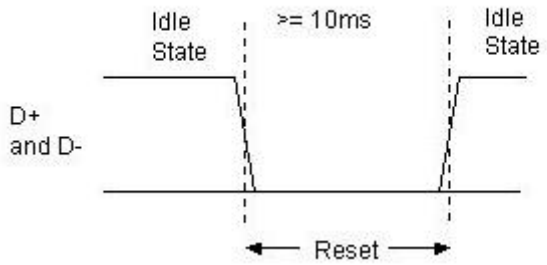


Ilustración 16: Señal de Reset

La señal de Start of Packet (SOP): es la señal utilizada para indicar el inicio de un paquete. Representa el primer bit del campo de sincronización (SYNC) de los paquetes. Esta señal es reconocida por el cambio de estado; del estado Idle (J) al estado opuesto (K).

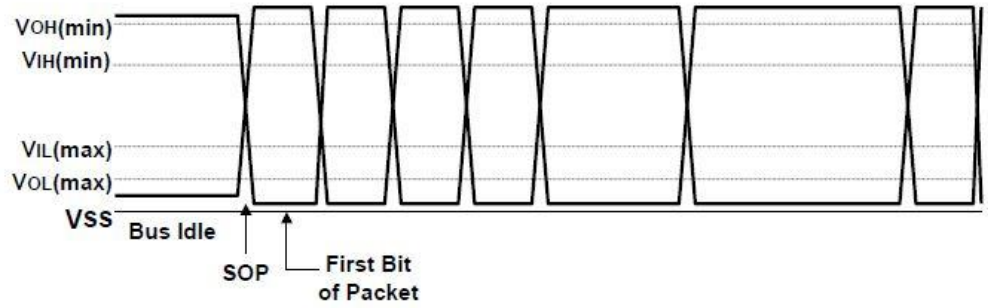


Ilustración 17: Start of packet

La señal de End of Packet (EOP): Esta señal es reconocida mediante el estado SE0 durante dos bits, seguido de un bit del estado Idle (J).

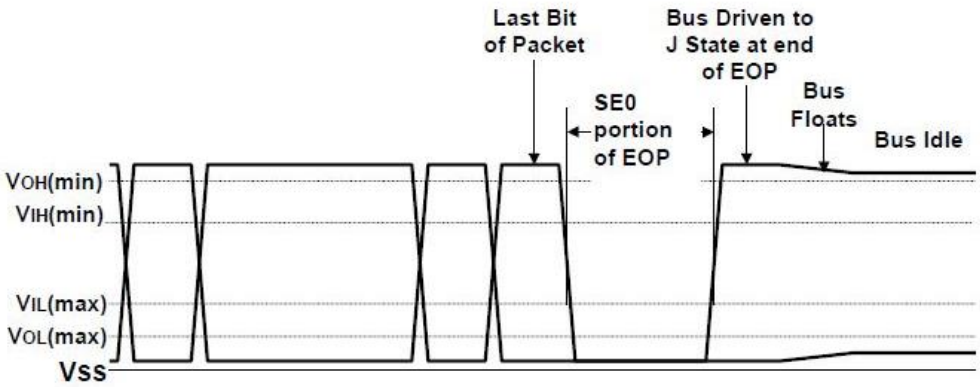


Ilustración 18: End of Packet

En la siguiente imagen se muestra un resumen de todos los estados y señales del bus.

Bus State	Levels
Differential '1'	D+ high, D- low
Differential '0'	D- high, D+ low
Single Ended Zero (SE0)	D+ and D- low
Single Ended One (SE1)	D+ and D- high
Data J State: Low-speed Full-speed	 Differential '0' Differential '1'
Data K State: Low-speed Full-speed	 Differential '1' Differential '0'
Idle State: Low-speed Full-speed	 D- high, D+- low D+ high, D- low
Resume State	Data K state
Start of Packet (SOP)	Data lines switch from idle to K state
End of Packet (EOP)	SE0 for 2 bit times followed by J state for 1 bit time
Disconnect	SE0 for >= 2us
Connect	Idle for 2.5us
Reset	SE0 for >= 2.5 us

Ilustración 19: Estados y señales del bus

Los dispositivos USB tienen la capacidad de entrar en un estado de bajo consumo cuando no se requiere su uso. En este estado los dispositivos no puede consumir más de 0.5 mA del bus, pero si está configurado como un dispositivo high power puede consumir hasta 2.5 mA. A este estado se le conoce como suspend mode.

Para entrar en el estado suspendido el dispositivo debe detectar que el bus está en el estado idle por más de 3 mili segundos como se muestra en la figura. Normalmente el host envía paquetes SOF (start of frame) cada 1 ms para así mantener el dispositivo despierto. Si se quiere que un dispositivo entre en este estado, simplemente no hay que mandar estos paquetes. En los dispositivos LS no es así, con estos se envía la señal EOP (estado SE0 durante dos bits) cada 1 ms para que no entren en el estado suspendido.

Un dispositivo en suspend mode debe de ser capaz de reconocer la señal de resume o de reset.

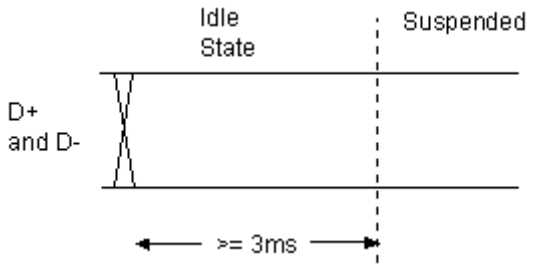


Ilustración 20: Señal de suspend mode

Para que un dispositivo salga del suspend mode, el host debe de cambiar el estado del bus de estado idle a estado (K) por lo menos durante 20 ms. Por último, envía un EOP para terminar el estado suspend mode.

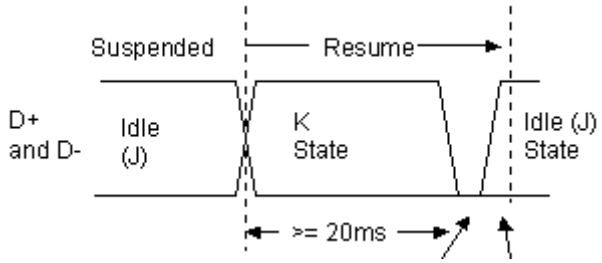


Ilustración 21: Señal fin suspend mode

Si un dispositivo quiere salir de este estado por su cuenta, primero deber de haber estado en suspend mode durante al menos 5 ms y después poner el bus en estado K entre 1 y 15 ms.

Un dispositivo USB indica su velocidad poniendo al conectarse D+ o D- a nivel alto, dependiendo del dispositivo que sea.

El dispositivo de FS que se muestra en la imagen tiene una pull-up resistor conectada a D+ que indica que tipo de dispositivo es. En la imagen de abajo se muestra un dispositivo LS con la resistencia pull-up en D-.

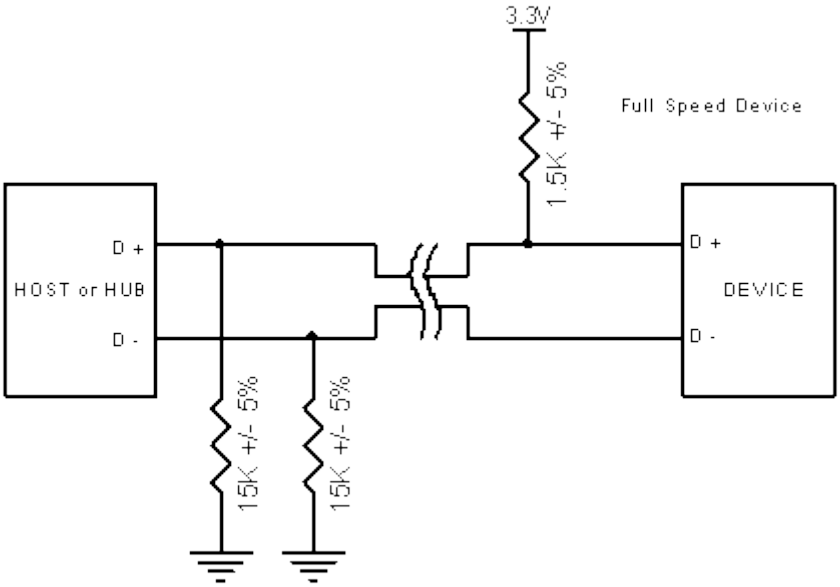


Ilustración 22: Conexión full speed

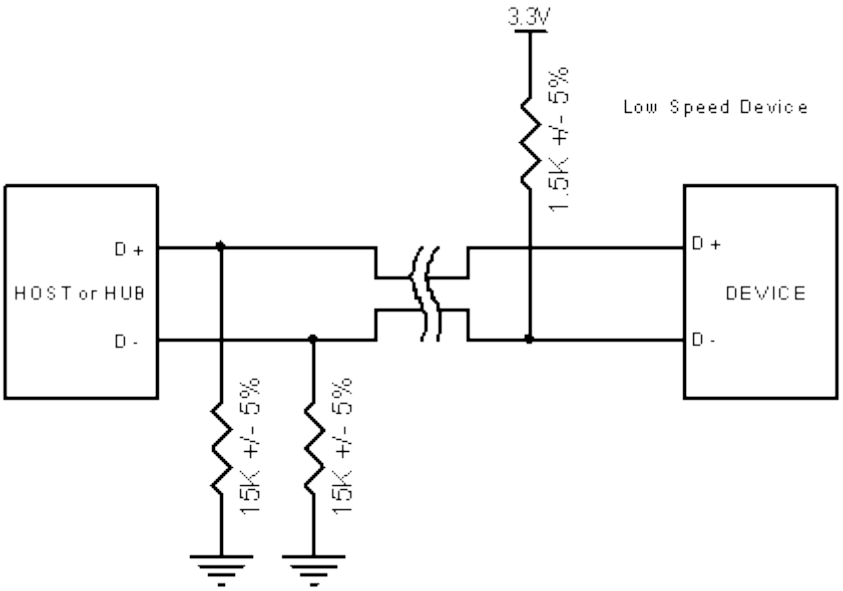


Ilustración 23: Conexión low speed

Para indicar que un dispositivo es HS, este, en primer lugar, se conecta como uno de FS con las pulled resistor puestas como se indica en la imagen de arriba. Una vez que ha sido conectado, durante el reset, hará la señal de que es un dispositivo HS y si el Hub lo permite, la conexión se hará en HS.

Por lo tanto, un dispositivo HS tiene que poder inicializarse por lo menos como un dispositivo de LS.

En la siguiente imagen se muestra los estados de la línea para establecer una conexión.

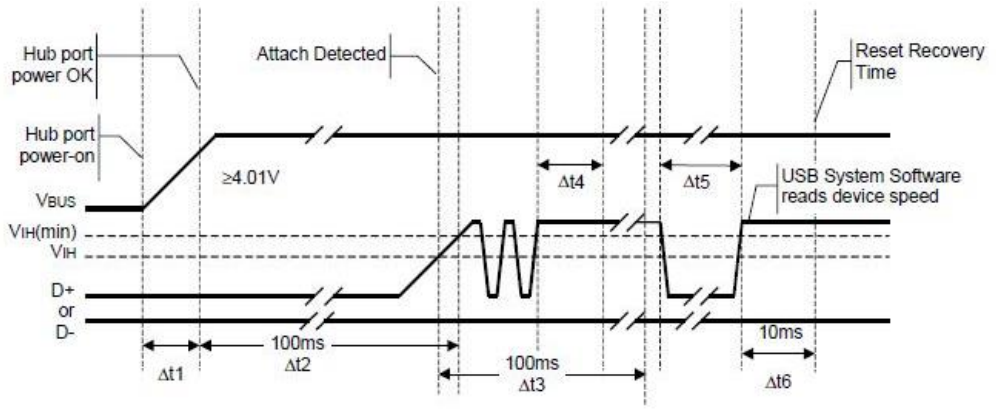


Ilustración 24: Pasos para la conexión

El **Δt1** corresponde al tiempo que requiere el Hub para dar potencia al dispositivo. Este tiempo está estipulado en el descriptor del Hub.

El **Δt2** corresponde al tiempo que requiere el dispositivo desde que Vbus está operativo hasta que envía la señal de conectado. Este tiempo debe ser menor a 100 ms.

El **Δt3** como poco ha de ser de 100 ms y es para asegurarse que la conexión es estable correctamente antes de enviar la señal de reset. Este intervalo empieza cuando al system software (ver apartado 2.2.4) se le notifica la señal de conexión.

En **Δt4** el bus está inactivo esperando el reset.

El **Δt5** corresponde a la señal de reset.

El **Δt6** dura 10 ms y da un tiempo de recuperación de la señal de reset.

[16] El host controla el tráfico de datos dividiendo el tiempo en intervalos llamados frames o microframes. Las frames son utilizadas para LS/FS y tiene un periodo de 1 ms mientras que las microframes están reservadas para HS y tiene un intervalo de tiempo de 125 micro segundos. Cada frame o microframe empieza con un paquete denominado SOF (star of frame).

En cada frame el host realiza las transmisiones para los diferentes dispositivos. En la siguiente imagen se ve un ejemplo de las frame en FS/LS ya que son por intervalos de 1 milisegundo.

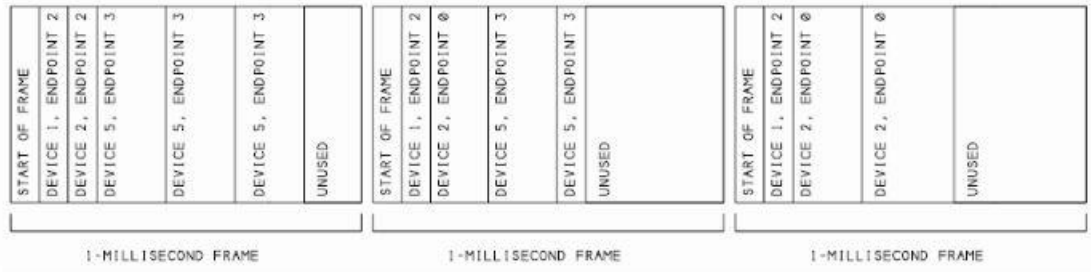


Ilustración 25: Representación de las frames

2.2.4 – PROTOCOLO DEL BUS

Como ya hemos dicho en la estructura USB hay un host, Hubs y funciones; el host básicamente se encarga de controlar el bus, de mandar y recibir información a las funciones y procesar la información.

El controlador del host se encarga de inicializar todas las transferencias de datos, tanto de entrada (transferencia IN) como de salida (transferencia OUT); la dirección de la transferencia siempre es con respecto al host

Todas las transacciones tienen tres paquetes; una transacción empieza cuando el host envía un paquete en el que está la información referida a la dirección de la transmisión (host-device o device-host), a la address (dirección) del dispositivo USB con el que se quiere comunicar , el tipo de mensaje que es y a que endpoint va dirigido. Este tipo de paquete se lo conoce como “token paket”.

Después el destinatario de la transmisión puede enviar la información o indicar que no tiene nada que enviar. Por último el destinatario en general envía un paquete handshake, indicando que la información ha sido transferida correctamente.

Algunas transacciones entre el Hub y el host tienen 4 paquetes; este tipo de transacciones es utilizado para la comunicación entre el host y dispositivos de full/low speed.

Por último, el modelo de transferencia de datos entre el host y el endpoint del destinatario se denomina pipe. Hay dos tipos:

- Stream pipes: que no tiene definida una estructura. Son utilizadas para las transmisiones bulk, control o interrupción (ver apartado2.2.4). Estas pipes pueden ser controladas tanto por el host como por los dispositivos.
- Mesagge pipes: que sí tiene definida una estructura, son utilizadas para las peticiones del host en las trasferencias de control y solo pueden ser controladas por el host.

Como mínimo una pipe (Default Control Pipe)) siempre debe de existir y una vez que está configurado el dispositivo, se crean nuevas pipes.

El sistema USB soporta que los dispositivos estén conectándose y desconectándose todo el tiempo, por lo que el sistema software tiene que estar acomodado a cambios dinámicos en la capa física.

Todos los dispositivos se conectan al sistema USB por los puertos que tienen los Hub. Todos los Hub tienen unos bits de estado que son usados para saber si un dispositivo está conectado o desconectado; el Host consulta estos bits y si detecta que hay un nuevo dispositivo conectado le asigna una única dirección y determina si el nuevo dispositivo conectado es un Hub o una función.

Si el dispositivo conectado es un Hub y hay dispositivos USB conectados a él, el procedimiento anterior es seguido para cada uno de los dispositivos. Si el dispositivo es una función, la notificación de que está conectado será manejada por el software del host de la manera apropiada para esa función.

Cuando un dispositivo ha sido quitado de un Hub, este inhabilita el puerto y le indica al host que ha sido quitado.

Esta indicación luego es manipulada apropiadamente por el sistema software USB. Si lo que se ha quitado es un Hub, es sistema software tiene

que ocuparse tanto del Hub como de todos los dispositivos que habían sido conectados al Hub anteriormente.

A continuación se describe el estado de los dispositivos:

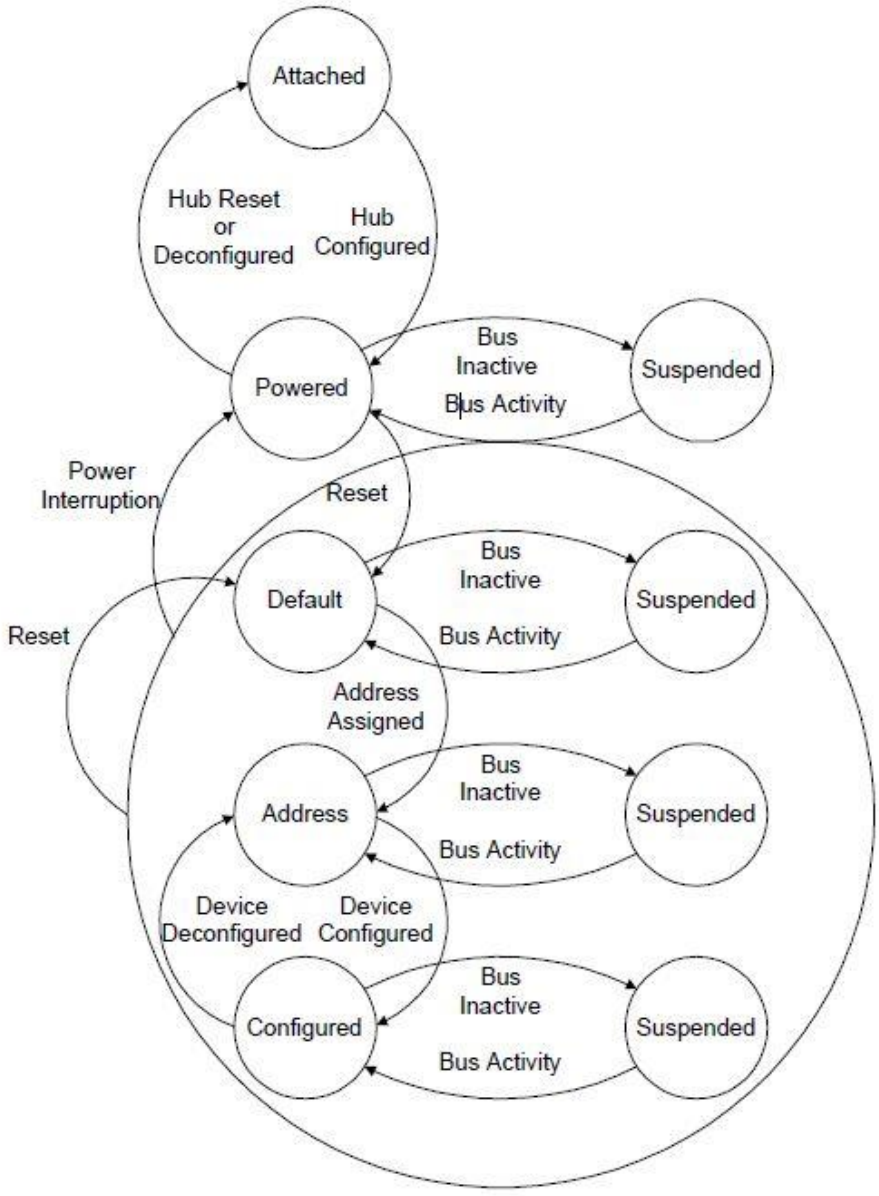


Ilustración 26: Estados de los dispositivos

Attach:

Al conectar un dispositivo a un puerto de un Hub, el dispositivo entra directamente en este estado.

Powered:

El dispositivo entra en este estado cuando es conectado al puerto de un Hub y recibe potencia de él, a no ser que sea un dispositivo que se alimenta por otros medios. Al entrar en este estado el Hub envía la señal de reset y se entra en el estado default.

Default:

Al recibir un reset el dispositivo entra en este estado. Al entrar en este estado la velocidad de comunicación ya se ha definido mediante las pulled resistor.

Address:

El dispositivo entra en este estado cuando el host ya le ha asignado su address mediante la default control pipe.

Configured:

Cuando al dispositivo ya se la ha asignado la address, el dispositivo se configura mediante la petición del host set configuration (ver apartado 2.2.6) enviada al endpoint cero. Una vez hecho esto el dispositivo ya está operativo para funcionar

Suspend:

Como se puede ver en todos los estados se puede entrar en el suspend mode si el bus está inactivo el tiempo necesario. Una vez asignada la address, a no ser que se reciba un reset, el dispositivo sigue manteniendo su dirección al entrar en este estado.

Para analizar el flujo de datos, en primer lugar hay que examinar 4 áreas implementadas en el sistema, que son las siguientes:

- **Dispositivos físicos USB:** la pieza de hardware al final del cable USB que representa algunas utilidades de la función. Éste a su vez se divide de otras tres áreas: función, USB logical device y USB bus interface.
- **Client software:** el software ejecutado en el host correspondiente a un dispositivo USB.

El client software pide que los datos sean movidos a través del USB entre un buffer en el host y un endpoint en los dispositivos USB. El host controller empaqueta los datos para moverlos a través del USB, y coordina el acceso al bus.
La siguiente figura demuestra la relación entre los endpoint, las pipes y los buffer del host.

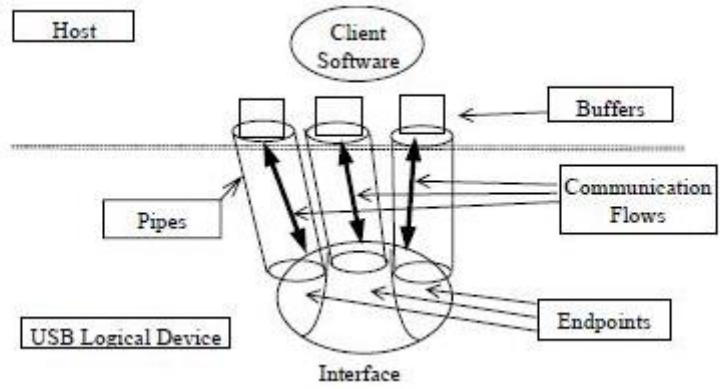


Ilustración 27: Relación pipes - endpoints

- **USB system software:** software que soporta el USB en un particular sistema de operación.
- **USB host controller:** hardware y software que permiten a los dispositivos USB estar conectados al host.

El host tiene dos interfaces:

- **Host controller driver (HCD):** la interface de software entre el USB host controller y el USB system software. Esta interface requiere que el controlador host tenga bastantes implementaciones sin el requerimiento de que todo el software del host sea dependiente de una implementación en particular.
- **USB driver:** la interface entre el USB system software y el client software.

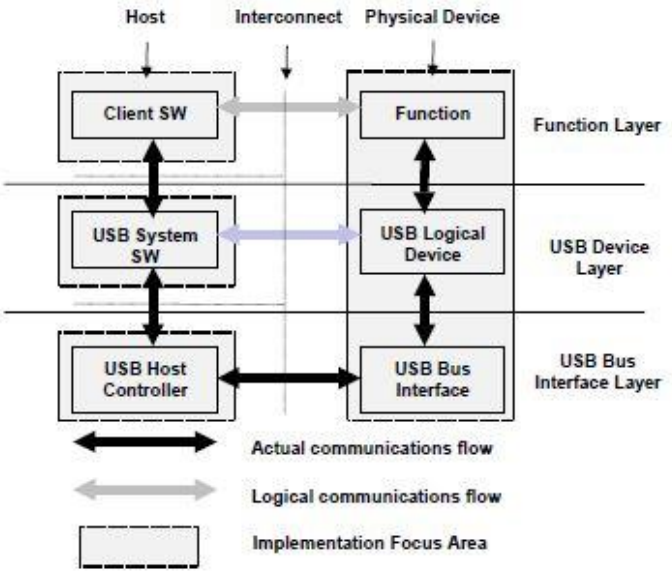


Ilustración 28: Representación flujo de datos

Como se muestra en la imagen anterior, una simple conexión entre un host y un dispositivo requiere la interacción entre múltiples capas e identidades.

- La capa USB bus interface proporciona la conexión física, de señal y de paquetes entre un host y un dispositivo.
- En la capa del dispositivo USB es donde el system software tiene una representación de las operaciones genéricas con un dispositivo.
- La capa de la función proporciona capacidades adicionales al host mediante una apropiada unión con la capa del client software.

El USB proporciona un servicio de comunicación entre el software en el host y una función. Las funciones pueden tener diferentes requerimientos de comunicación para las diferentes interacciones entre cliente y función. Cada flujo de comunicación se termina en un endpoint de un dispositivo, que son utilizados para identificar aspectos de los flujos de comunicación.

Por otro lado, un dispositivo lógico USB aparece en el sistema como una colección de endpoint. Los endpoint son agrupados e implementados como una interface.

Mientras que los dispositivos físicamente están conectados al USB en una estructura en estrella por niveles, la comunicación del host con cada

dispositivo lógico es como si estuvieran directamente conectados al puerto Root Hub. En la siguiente figura se muestra esta estructura de conexión lógica. Los Hubs, aparte de funciones, son también dispositivos lógicos. Cuando un Hub es quitado, todos los dispositivos conectados al host deben de ser quitados de la visión del host de esta estructura lógica.

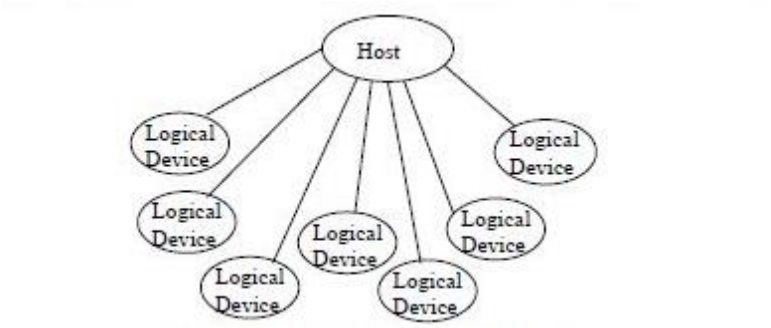


Ilustración 29: Topología lógica del bus

Un endpoint es una porción identificable de un dispositivo USB, supone el final del flujo de comunicación entre el host y el dispositivo. Cada dispositivo lógico USB está compuesto de una colección de endpoints independientes, al igual que cada dispositivo tiene una única dirección asignado cuando se conecta al sistema, a cada endpoint se le asigna un único identificador llamado el endpoint name. Cada endpoint tiene determinado la dirección del flujo de datos.

La combinación de la dirección del dispositivo (address), el número del endpoint y la dirección del flujo de datos permite que cada endpoint sea debidamente identificado. Cada endpoint es una simple conexión que soporta el flujo de datos en una dirección, IN (del dispositivo al host) o OUT (del host al dispositivo).

Las características de un endpoint determinan los requerimientos para las transferencias. Un endpoint se describe a sí mismo con:

- Los requerimientos de frecuencia de acceso al bus
- Los requerimientos de la bandwidth.
- El número de endpoint.
- El tamaño máximo de los paquetes que es capaz de enviar o recibir.
- La dirección en que los datos son transmitidos.

Para configurar los dispositivos USB se usan ambos endpoint, el de entrada y el de salida, del endpoint número 0. El USB system software utiliza el método de control por defecto para inicializar y manipular el dispositivo lógico mediante la default control pipe. Esta pipe provee de acceso a la información del dispositivo.

El endpoint número 0 siempre es accesible una vez que el dispositivo está conectado, encendido y ha recibido un bus reset.

Un dispositivo que opera a high speed debe poder soportar mínimamente operaciones a full speed para poder configurarse.

[3] El USB realiza la transacción de datos mediante unos paquetes ya definidos. Los paquetes son los elementos de los que se compone una transacción. Antes y después de un paquete el estado del bus es idle.

El formato de un paquete es como se muestra en la figura, primero el SYNC después los datos y por último el EOP.

Ilustración 30: Formato de los paquetes

El primer byte después del SYNC corresponde al PID. Todos los paquetes tienen un PID que identifica que tipo de paquete es. El PID solo necesita 4 bits, los otros 4 son repetidos de una forma complementaria. En la siguiente imagen se ve el valor del PID dependiendo del tipo de paquete que sea.

PID Type	PID Name	PID<3:0>*
Token	OUT	0001b
	IN	1001b
	SOF	0101b
	SETUP	1101b
Data	DATA0	0011b
	DATA1	1011b
	DATA2	0111b
	MDATA	1111b
Handshake	ACK	0010b
	NAK	1010b
	STALL	1110b
	NYET	0110b

Ilustración 31: Tipos de PID

Token packet: son los primeros paquetes en todas las transacciones. En ellos va incluida la address y el endpoint. Además, estos paquetes como los data paket incluyen el CRC (cyclic redundance code) que mediante un algoritmo devuelve un número que es usado para verificar que los datos se han recibido correctamente. USB utiliza dos CRC diferentes dependiendo del paquete, uno de 5 bits (para los token pakets) y otra de 16 bits (para los data pakets)

Como se ve en la imagen hay 4 tipos de paquetes token.

- **OUT:** Indican que la transmisión es del host al dispositivo
- **IN:** indican que la transmisión es del dispositivo al host
- **SETUP:** indican que es una transferencia de control.
- **SOF:** la información de estos paquetes está más adelante.

Con un paquete token por lo tanto se indica a que dispositivo va dirigida la transacción mediante la address, a que endpoint mediante el endpoint number y el tipo de transacción de que se trata indicando el tipo de PID.

Sync	PID	ADDR	ENDP	CRC5	EOP
	8 bits	7 bits	4 bits	5 bits	

Ilustración 32: Formato Token packet

Data packet: en él va la información que se quiere transmitir. Como se ve en la imagen siguiente hay 4 tipos de paquetes de datos. En FS/LS y dos tipos de paquetes de datos: Data0 y Data1 que se van alternando entre transacciones y funciona como un chequeo de errores adicional. En HS también están los paquetes DATA2 y DATAM.

En dispositivos LS el tamaño máximo de los datos que se puede enviar es de 8 bytes, mientras que en FS es de 1023 y en HS es de 1024 bytes.

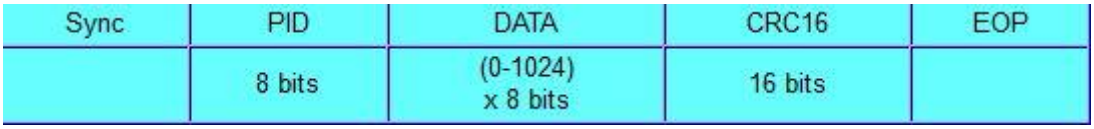


Ilustración 33: Formato Data packet

Handshake packets: solo constan del PID y son utilizados para saber el estado de la transacción. Como se ve en la imagen anterior hay 4 tipos:

ACK: Verifica que la transacción ha sido realizada sin errores

NAK: Informa de que el dispositivo no puede ni enviar ni recibir datos porque está ocupado o que no tiene nada para enviar.

STALL: informa de que el endpoint no está operativo o que no se puede aceptar la pipe de control.

NYTE: utilizado en HS solamente.

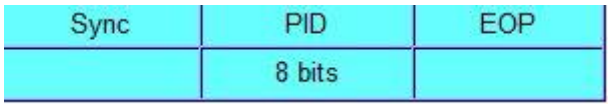


Ilustración 34: Formato Handshake packet

SOF packets: la utilidad de estos paquetes se ha citado en el apartado 2.3.3 al hablar de las frames y microframes. Utiliza un CRC de 5 bits. Lo forma de estos paquetes es la siguiente:

Sync	PID	Frame No.	CRC5	EOP
	8 bits	11 bits	5 bits	

Ilustración 35: Formato SOF packet

El sistema USB se comunica por medio de transacciones. Una transacción es una secuencia de tres paquetes, un token packet, un data packet y un handshake packet. En las transmisiones isócronas de las que se habla más adelante solo hay dos paquetes ya que el handshake se omite al no ser necesario el chequeo de errores.

Hay tres tipos de transacciones: transacciones IN, Transacciones OUT y Transacciones SETUP.

Las transacciones IN: son utilizadas para enviar datos de un dispositivo al host.

En la siguiente imagen se muestra la forma de las transacciones IN. Lo azul representa lo enviado por el host y lo blanco lo enviado por el dispositivo. El data x es porque el PID puede ser data 1 o data 0.

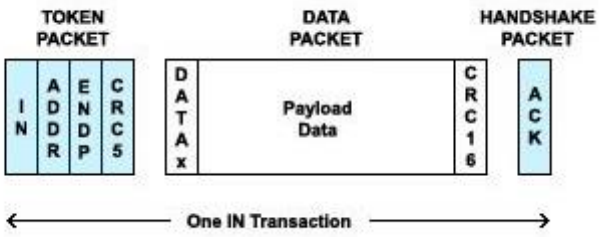


Ilustración 36: Transacción IN

Primero el host envía un paquete token, luego el dispositivo envía los datos y por último el host, después de comprobar el CRC envía el ACK confirmando que el paquete ha sido enviado correctamente.

En la siguiente imagen se muestra una transacción IN con un Hub entre el dispositivo y el host. Como se puede ver hay un paquete del que no se ha hablado entre el host y el Hub, eso es debido a que la comunicación entre el Hub y el host es a HS y esta comunicación utiliza otro tipo de paquete especial del que no se quiere entrar en detalle en este trabajo. La comunicación entre el Hub y el dispositivo no necesita este paquete adicional ya que se supone que es FS/LS.

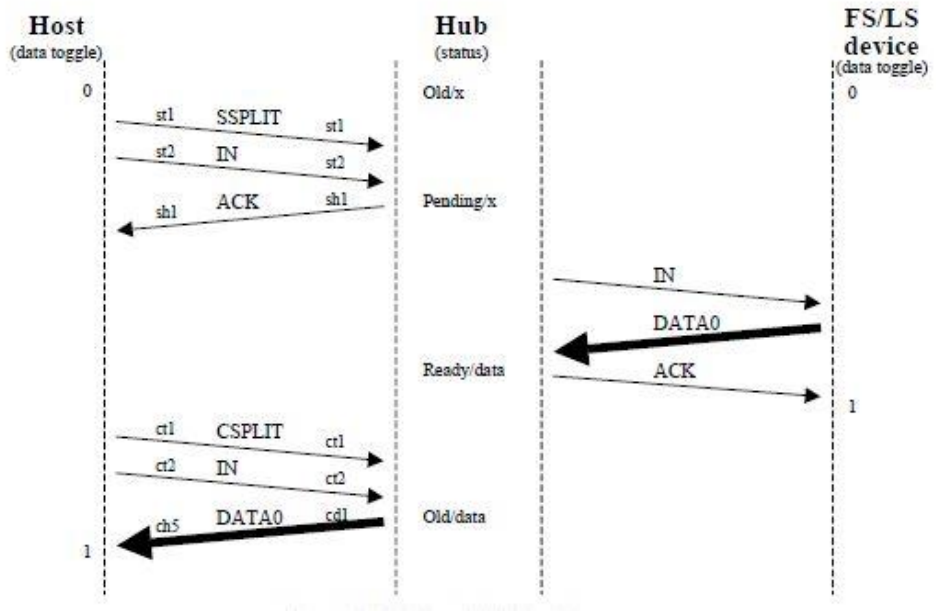


Ilustración 37: Transacción IN con HUB

Las transacciones OUT: son utilizadas para enviar datos del host al dispositivo. En la siguiente imagen se muestra la forma de las transacciones OUT.

Primero el host envía un paquete token, luego envía los datos y por último el dispositivo, después de comprobar el CRC envía el paquete ACK confirmando que el paquete ha sido enviado correctamente.

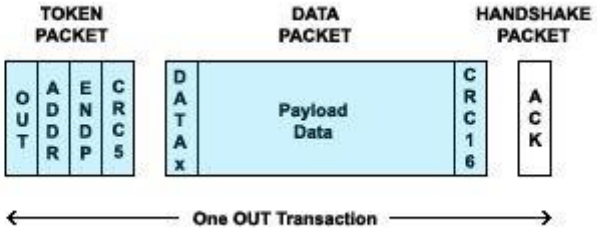


Ilustración 38: Transacción OUT

En la siguiente imagen se muestra una transacción OUT con un HUB entre el dispositivo y el host. Como se puede ver, la forma de este tipo de transacciones es la misma que las transacciones setup.

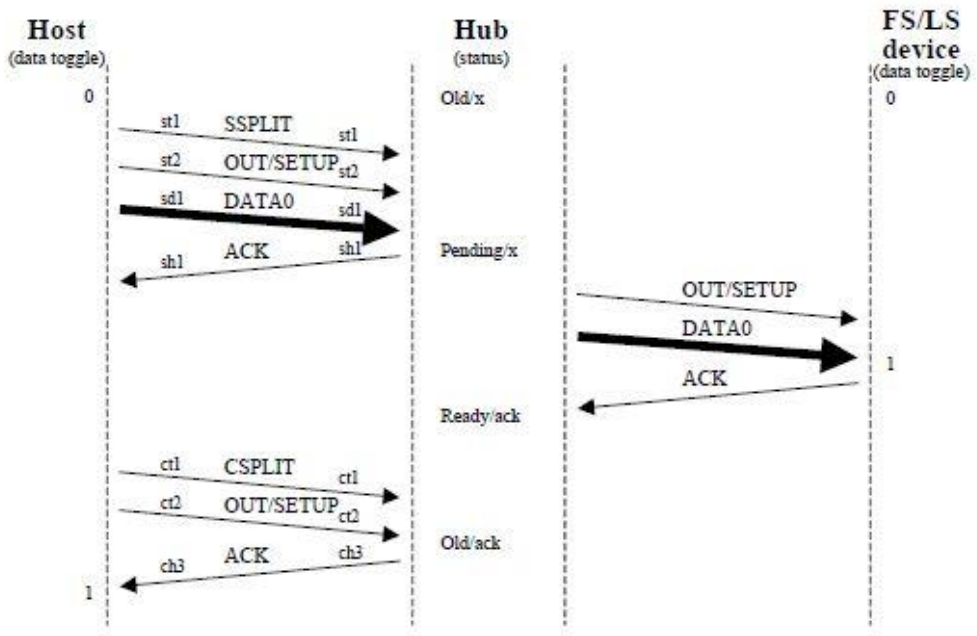


Ilustración 39: Transacción OUT con HUB

Las transacciones SETUP: son muy parecidas a las transacciones OUT con la diferencia de que el tamaño del paquete de datos (sin contar el PID y el CRC) es de exactamente 8 bytes. Este tipo de transacciones son las

primeras utilizadas en las transmisiones de control explicadas más adelante. El PID del paquete de datos de estas transacciones siempre es el DATA0. A continuación se muestra su forma.

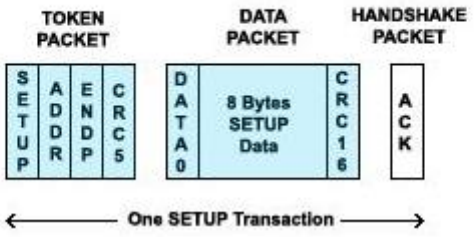


Ilustración 40: Transacción SETUP

Hay 4 maneras diferentes de transmitir datos por USB, cada una tiene sus propias características y objetivos. Están construidas utilizando una o varias transacciones.

Las **transmisiones de control:** son las primeras que recibe un dispositivo y se encargan de configurar el dispositivo, obtener información sobre el dispositivo, el envío de comandos al dispositivo, o recuperar los informes sobre la situación del dispositivo.

Una transmisión de control tiene tres etapas

- **Setup Stage**:

  Es una transacción de tipo SETUP y es donde la petición del host es enviada (de las peticiones del paquete de datos de setup se habla en el apartado 2.2.6.).

  Como se muestra la imagen, si el paquete es recibido correctamente por el dispositivo éste envía el paquete ACK, en caso contrario, no envía nada.

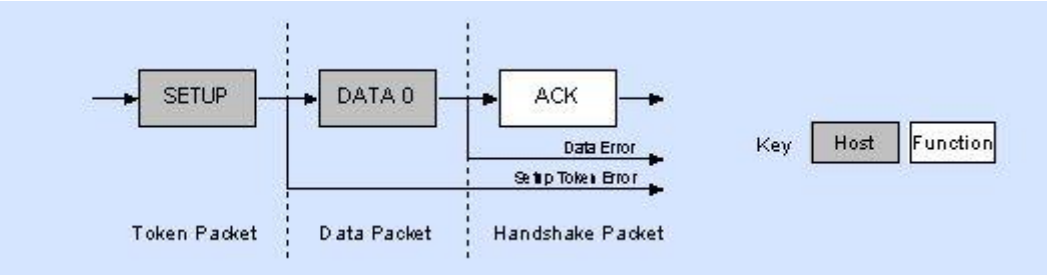


Ilustración 41: Representación setup stage

- **Data Stage**:

  En esta etapa los datos pedidos en la petición de la etapa anterior son enviados. En el caso de que el número de datos exceda del máximo habrá más de una transacción. En esta etapa, dependiendo de la petición del host, puede haber transacciones de tipo IN(si el host quiere recibir información) o del tipo OUT (si el host quiere enviar información). En la siguiente imagen se muestra la forma de las transacciones y los correspondientes handshake que se envían dependiendo de la situación.

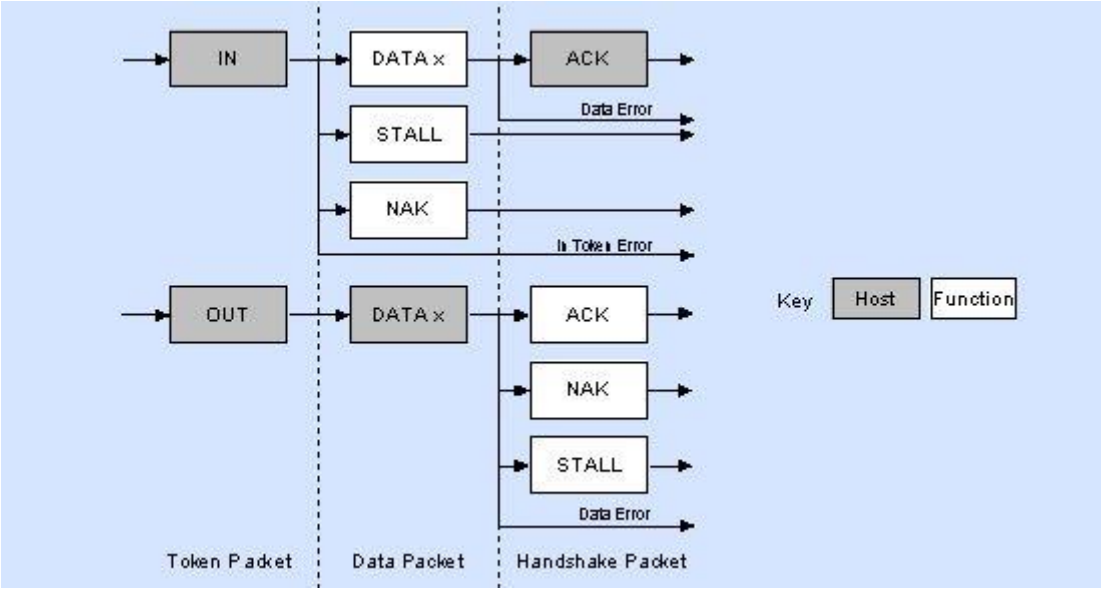


Ilustración 42: Representación data stage

- **Status Stage**:

  Esta etapa sirve para confirmas las peticiones del host.

  Si en la Data stage el host ha enviado una transacción IN (es decir quería recibir datos), en esta etapa envía una transacción OUT con cero bits en el paquete de datos (exceptuando el PID y el CRC), si el dispositivo ha recibido y procesado las peticiones correctamente envía el ACK, en caso contrario, envía el paquete handshake correspondiente.

  Si por el contrario, el host en la etapa anterior ha enviado una transacción OUT (quería enviar datos) en esta etapa envía una transacción IN, el dispositivo al recibir el paquete token envía un paquete de datos con cero bits y por último el host confirma el recibo del paquete con un ACK. Si hay algún error al recibir el paquete token, el dispositivo envía el correspondiente handshake.

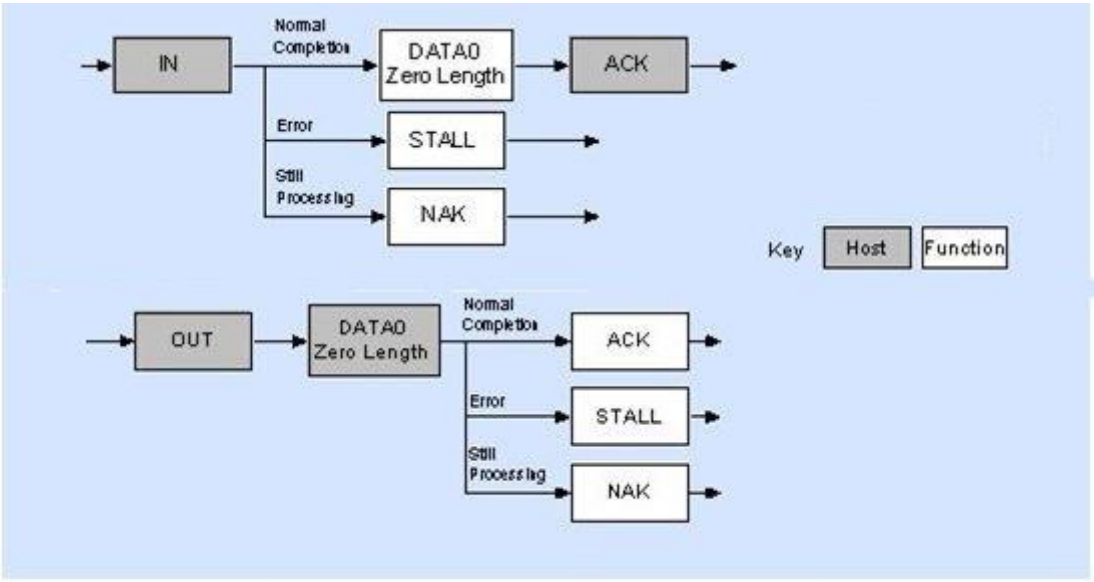


Ilustración 43: Representación status stage

Transmisiones de Interrupción: aunque tengan este nombre, las transmisiones de interrupción no tienen nada que ver con las interrupciones. Llevan este nombre porque están asociadas a dispositivos que anteriormente si utilizaban interrupciones como un ratón o un teclado. Pero como hemos dicho, bajo el sistema USB si un dispositivo requiere la

atención del host, éste tiene que esperar hasta que el host le pida que envíe sus datos. Este tipo de trasmisiones son utilizadas para enviar y recibir pequeñas cantidades de datos y está garantizado el ancho de banda que el dispositivo necesita. El ratio de polling (las veces que el host le pide al dispositivo que envíe sus datos) está definido en la descripción del dispositivo al configurarse.

En este tipo de transmisiones se utilizan las transacciones IN como OUT. En la siguiente imagen se muestran ambas transacciones y los handshake que se envían dependiendo de la situación.

En estas transmisiones el tamaño máximo de los paquetes de datos es de 1 a 8 bytes para low speed, de 1 a 64 para full speed y por encima de 1024 para dispositivos de alta velocidad.

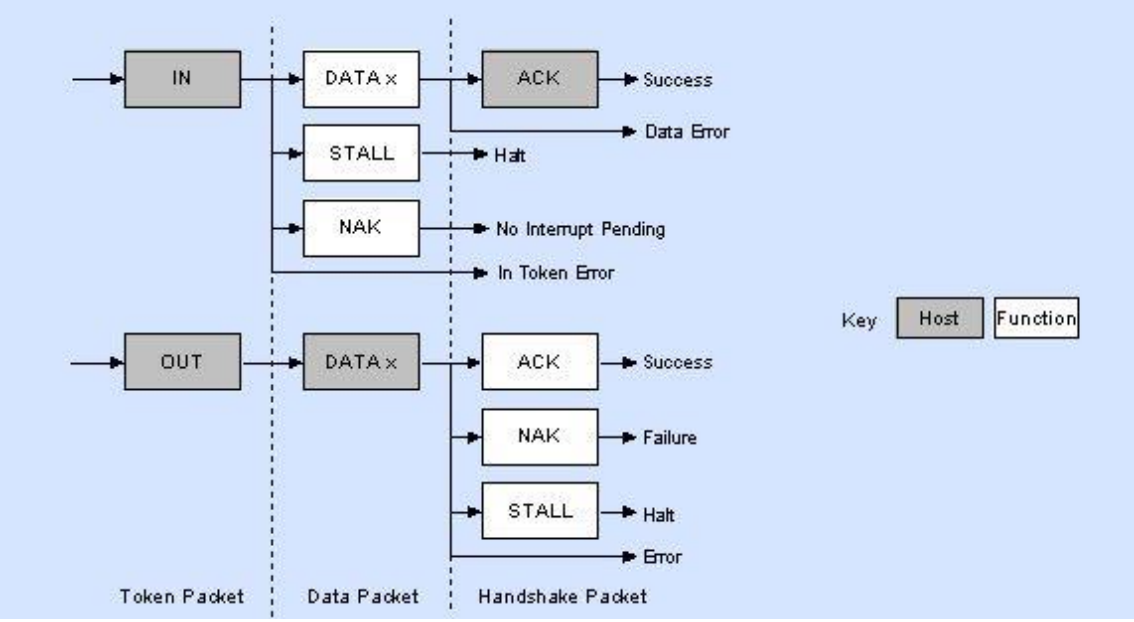


Ilustración 44: Representación transmisión de interrupción

Las **Transmisiones Bulk**: son utilizadas para transferir gran cantidad de datos. El problema de esta transferencia es que el ancho de banda no está garantizado, es decir que si no hay suficiente espacio en el bus, los datos se envían a través de varias transferencias. Por lo tanto, este tipo de transferencias es utilizado cuando se quiere enviar gran cantidad de datos lo más rápido posible y sin importar que haya una interrupción en el envío de

los mismos, como por ejemplo en una impresora y dispositivos de almacenamiento. Este tipo de transmisiones utiliza transacciones tanto OUT como IN. Los dispositivos de tipo LS no pueden utilizar este tipo de transmisiones. En la siguiente imagen se representa esta transmisión y los Handshake correspondientes dependiendo de la situación.

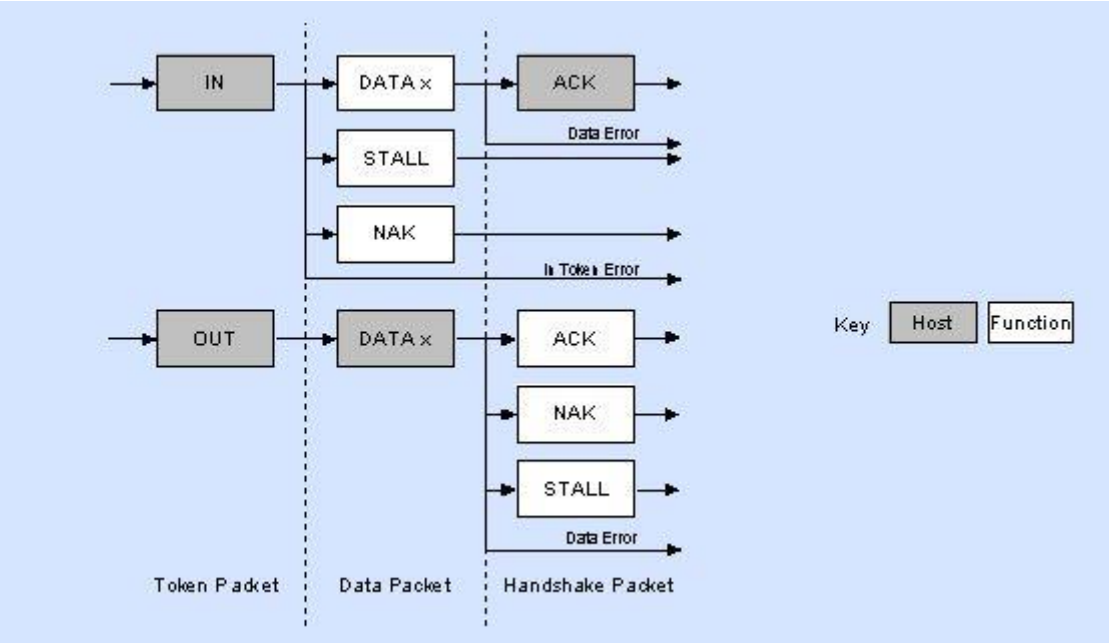


Ilustración 45: Representación transmisión bulk

Las **transmisiones Isócronas**: son usadas para la transferencia de grandes cantidades de datos pero sin confirmación de errores, por lo que la llegada correcta de estos datos no está garantizada. Es usada en dispositivos que puedan permitirse perder datos pero que necesiten un flujo constante de datos, como podría ser dispositivos de audio y de video. La transferencia isócrona provee comunicación continua y periódica entre el host y el dispositivo. En estas transmisiones el ancho de banda está garantizado. En la siguiente imagen se muestra las transmisiones isócronas y como se puede apreciar no hay handshake ya que no hay confirmación de recibimiento correcto de los paquetes.

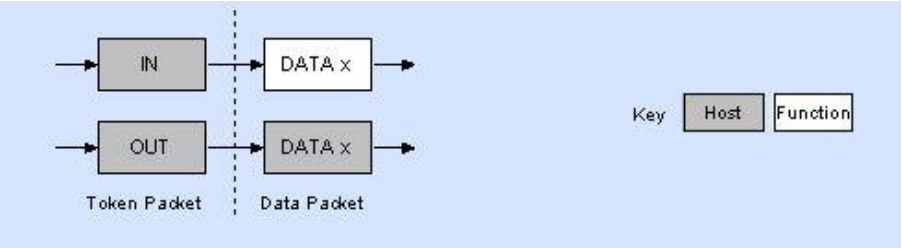


Ilustración 46: Representación transmisión isócrona

2.2.5 – DESCRIPTORES USB

[8] Todos los dispositivos USB tienen una serie de descriptores de los que se hablará a continuación, que son necesarios para que el host pueda saber qué tipo de dispositivo es, cuantas configuraciones soporta, el número de los endpoint y de qué tipo son (para transferencias bulk, isócronas o de interrupción).

Los descriptores más importantes son los device descriptors, configuration descriptors, interface descriptors y endpoint descriptors.

En la siguiente imagen se muestra un esquema en el que se muestra la manera en la que un dispositivo es descrito.

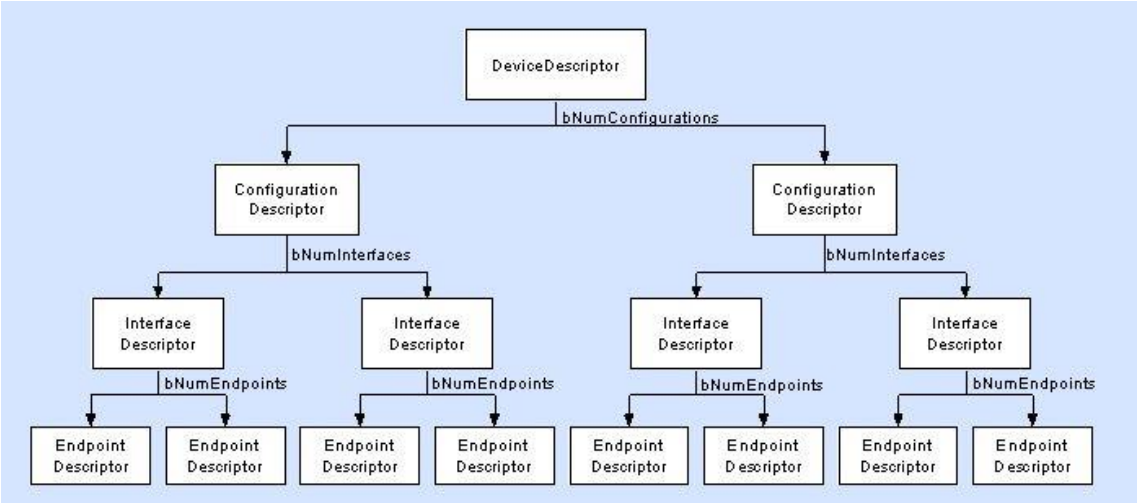


Ilustración 47: Esquema descriptores USB

Como se ve en la imagen, un dispositivo solo puede tener un device descriptor, que incluye información tal como la versión de USB que soporta el dispositivo, el product y el vendor ID o el número de posible configuraciones que el dispositivo puede tener.

Un dispositivo, como se ve en la imagen, puede tener más de una configuración diferente, en la que se especifica la cantidad de potencia que va a utilizar en esta configuración (mediante el Vbus) o si se alimenta mediante otros medios. También se especifica el número de interfaces que tiene esta configuración en concreto. Solo puede estar operativa una configuración a la vez; el host, una vez ha enumerado el dispositivo y ha leído los descriptores, toma la decisión de que configuración utilizar. Para cambiar la configuración, el dispositivo no debe tener ninguna actividad en los endpoints.

El interface descriptor se puede ver como un grupo de endpoints que forman un dispositivo con unas determinadas características. Al contrario de las configuraciones, sí que puede haber más de una interface funcionando a la vez.

El endpoint descriptor sirve para especificar el tipo de transmisión que soporta el endpoint junto con la dirección, el tamaño máximo de los paquetes o la frecuencia de polling (del host) que necesita. El EP0 se asume que es un endpoint de control por lo que no necesita descriptor.

Todos los descriptores tienen el campo bLenght que sirve para saber el tamaño del descriptor, si el descriptor es mayor que lo indicado aquí el host ignorará los restantes bytes. También tienen el campo bDescriptorType que indica que tipo de descriptor es (de dispositivo, de configuración etc).

En la siguiente imagen se muestra de que campos está compuesto el device descriptor.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of the Descriptor in Bytes (18 bytes)
1	bDescriptorType	1	Constant	Device Descriptor (0x01)
2	bcdUSB	2	BCD	USB Specification Number which device complies too.
4	bDeviceClass	1	Class	Class Code (Assigned by USB Org) If equal to Zero, each interface specifies it's own class code If equal to 0xFF, the class code is vendor specified. Otherwise field is valid Class Code.
5	bDeviceSubClass	1	SubClass	Subclass Code (Assigned by USB Org)
6	bDeviceProtocol	1	Protocol	Protocol Code (Assigned by USB Org)
7	bMaxPacketSize	1	Number	Maximum Packet Size for Zero Endpoint. Valid Sizes are 8, 16, 32, 64
8	idVendor	2	ID	Vendor ID (Assigned by USB Org)
10	idProduct	2	ID	Product ID (Assigned by Manufacturer)
12	bcdDevice	2	BCD	Device Release Number
14	iManufacturer	1	Index	Index of Manufacturer String Descriptor
15	iProduct	1	Index	Index of Product String Descriptor
16	iSerialNumber	1	Index	Index of Serial Number String Descriptor
17	bNumConfigurations	1	Integer	Number of Possible Configurations

Ilustración 48: Device descriptors

Los campos del Configure Descriptor se muestran a continuación.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	Configuration Descriptor (0x02)
2	wTotalLength	2	Number	Total length in bytes of data returned
4	bNumInterfaces	1	Number	Number of Interfaces
5	bConfigurationValue	1	Number	Value to use as an argument to select this configuration
6	iConfiguration	1	Index	Index of String Descriptor describing this configuration
7	bmAttributes	1	Bitmap	D7 Reserved, set to 1. (USB 1.0 Bus Powered) D6 Self Powered D5 Remote Wakeup D4..0 Reserved, set to 0.
8	bMaxPower	1	mA	Maximum Power Consumption in 2mA units

Ilustración 49: Configure descriptors

El Interface Descriptor tiene el campo bInterfacenumber que especifica el número de interfaz y también el campo bAlternateSetting que permite a la interfaz cambiar la configuración a una alternativa. Es decir, inicialmente se puede trabajar con "x" interfaces con su número correspondiente y a la vez haber descrito una interfaz más con un número que esté dentro del valor "x" pero con bAlternateSetting puesto en "1". Inicialmente, los dispositivos utilizarán las interfaces por defecto, es decir las que tienen el bAlternateSetting con un valor 0. Si en algún momento el host quiere cambiar de interfaz, mediante la petición Set Interface (ver apartado 2.2.6), puede cambiar la configuración de una interfaz por defecto por su alternativa, que tiene el descriptor bAlternateSetting en 1. En la siguiente imagen se muestra la composición de este descriptor.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes (9 Bytes)
1	bDescriptorType	1	Constant	Interface Descriptor (0x04)
2	bInterfaceNumber	1	Number	Number of Interface
3	bAlternateSetting	1	Number	Value used to select alternative setting
4	bNumEndpoints	1	Number	Number of Endpoints used for this interface
5	bInterfaceClass	1	Class	Class Code (Assigned by USB Org)
6	bInterfaceSubClass	1	SubClass	Subclass Code (Assigned by USB Org)
7	bInterfaceProtocol	1	Protocol	Protocol Code (Assigned by USB Org)
8	iInterface	1	Index	Index of String Descriptor Describing this interface

Ilustración 50: Interface descriptors

El endpoint Descriptor es usado para describir los endpoint distintos del EP0 ya que éste ya está definido por defecto. En la imagen siguiente se muestra los diferentes campos de este descriptor.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes (7 bytes)
1	bDescriptorType	1	Constant	Endpoint Descriptor (0x05)
2	bEndpointAddress	1	Endpoint	Endpoint Address Bits 0..3b Endpoint Number. Bits 4..6b Reserved. Set to Zero Bits 7 Direction 0 = Out, 1 = In (Ignored for Control Endpoints)
3	bmAttributes	1	Bitmap	Bits 0..1 Transfer Type 00 = Control 01 = Isochronous 10 = Bulk 11 = Interrupt Bits 2..7 are reserved. If Isochronous endpoint, Bits 3..2 = Synchronisation Type (Iso Mode) 00 = No Synchonisation 01 = Asynchronous 10 = Adaptive 11 = Synchronous Bits 5..4 = Usage Type (Iso Mode) 00 = Data Endpoint 01 = Feedback Endpoint 10 = Explicit Feedback Data Endpoint 11 = Reserved
4	wMaxPacketSize	2	Number	Maximum Packet Size this endpoint is capable of sending or receiving
6	bInterval	1	Number	Interval for polling endpoint data transfers. Value in frame counts. Ignored for Bulk & Control Endpoints. Isochronous must equal 1 and field may range from 1 to 255 for interrupt endpoints.

Ilustración 51: Endpoint descriptors

2.2.6 - PETICIONES DEL HOST

Como ya se ha indicado, los dispositivos tienen que ser capaces de responder a los paquetes de setup del host transferidos en la default control pipe en el EP0. Estos paquetes son necesarios para obtener la información del dispositivo o asignarle la address. La forma de estos paquetes es la siguiente:

Offset	Field	Size	Value	Description
0	bmRequestType	1	Bitmap	D7 Data direction 0 - Host-to-device 1 - Device-to-host D6:5 Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4:0 Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4-31 = Reserved
1	bRequest	1	Value	Specific Request
2	wValue	2	Value	Use varies according to request
4	wIndex	2	Index or Offset	Use varies according to request
6	wLength	2	Count	Number of bytes to transfer if there is a data stage

Ilustración 52: Formato del setup packet

El bmRequestType determina la dirección de la petición, el tipo y a que recipiente va asignado. El campo bRequest determina la petición que se está haciendo. Por otro lado wValue y Windex varían dependiendo de la petición. Por último wLenght informa del tamaño del paquete de datos de la stage data, si hubiera dicha etapa (no siempre es necesaria).

En la siguiente tabla se muestra todas las peticiones estándar que se pueden hacer a los dispositivos. El apartado de data se refiere a los datos que se envía en la stage data si fuese necesaria esta etapa.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Interface
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
00000001B	SET_INTERFACE	Alternate Setting	Interface	Zero	None
10000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

Ilustración 53: Peticiones estándar del host

- **Get configuración:** Con esta petición el dispositivo devuelve la configuración, si no está configurado devuelve el valor 0.

- **Set configuración**: con esta petición el host pone la configuración que quiere de los descriptores del dispositivo.

- **Get descriptor:** Esta petición devuelve los descriptores del recipiente si los hubiese.

- **Set descriptor:** Con esta petición el host puede añadir nuevos descriptores al dispositivo.

- **Get status**: devuelve el estado del recipiente seleccionado.

- **Get interface:** devuelve la interfaz que se está utilizando.

- **Set interface:** El host pone una configuración alternativa de la interfaz que se está utilizando, previamente descrita en los descriptores del dispositivo.

- **Set address:** con esta petición el host pone la address al dispositivo.

- **Clear /set features:** sirve para deshabilitar o poner una característica concreta .

- **Synch frame:** sirve para reportar la sincronización de la frame de los endpoints.

2.3 - MBED

2.3.1 - INTRODUCCIÓN

[2] La MBED es una plataforma utilizada para el desarrollo de pequeños dispositivos basados en microcontroladores ARM Cortex-M de 32 bits; la relativa simplicidad de la arquitectura de estos microcontroladores los hace ideales para aplicaciones de baja potencia, por ello dominan el mercado en lo referente a la electrónica móvil e integrada.

La MBED es un proyecto desarrollado por la empresa ARM. Estas plataformas están diseñadas para el fácil y rápido desarrollo de productos, enfocado especialmente en conectar dispositivos "Internet of things"; es decir que los dispositivos estén conectados a internet.

La plataforma MBED provee gratis los diseños de hardware, librerías y herramientas online (como el compilador) para un prototipado rápido de productos basados en microcontroladores ARM.

2.3.2 - MBED LPC 1768.

Esta MBED es llamada así por el microcontrolador LCP 1768 del que se habla más adelante. Está diseñada para el prototipado de pequeños dispositivos.

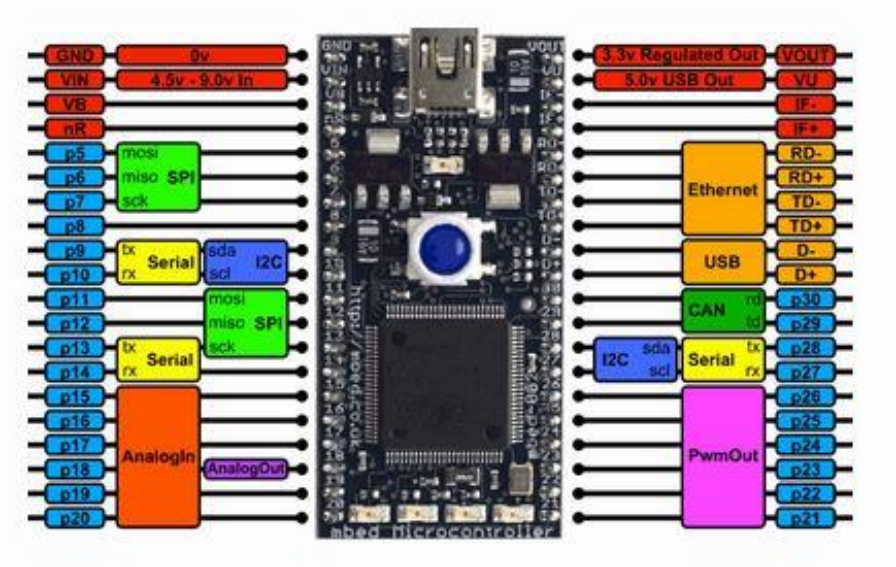


Ilustración 54: MBED LCP1768

Como se ve en la imagen, la MBED posibilita la comunicación con múltiples dispositivos a través de múltiples protocolos de comunicación. La elección de esta MBED en particular se debe básicamente a dos razones, la primera es su tamaño, ya que había que incorporarlo al Robot TEO, y la segunda por sus líneas SPI, USB necesarias para comunicarse como se describe en el apartado 2.1.

Aparte de estas características la MBED posee múltiples pines digitales tanto de entrada como de salida, asi como 6 pines para poder enviar señales PWM y 4 leds programables.

Para alimentar la MBED se puede hacer mediante el pin Vin (de 4.5V a 9 V) y GND o mediante el conector USB. También lleva incorporado un botón de reset, necesario a la hora de poner un nuevo programa.

2.3.3 - COMUNICACIÓN CON LA MBED.

A continuación se tratará sobre todo lo relacionado con la comunicación MBED mediante el protocolo USB.

Para ello primero se hablará del microcontrolador LCP 1768 y en especial de la parte del microcontrolador relacionada con el controlador del dispositivo USB. En segundo lugar, de todas las librerías necesarias para

configurar el dispositivo como un dispositivo USB, más concretamente como un dispositivo HID (human interface device).

[19] Este microcontrolador, como ya se ha dicho, está basado en ARM Cortex-M. La CPU tiene una velocidad de operación de 100 MHz. Algunas de sus características son: una memoria flash de 512 KB, 32 KB de RAM, diferentes interfaces para poder comunicarse mediante Ethernet, USB (utilizando el microcontrolador como un Host, un dispositivo o como OTG [host o dispositivo dependiendo de la situación]), posee dos canales CAN, una interfaz SPI, 3 interfaces I2C, múltiples pines digitales I/O, señales de PWM etc. Se alimenta a 3.3 V.

En la siguiente imagen se ve el diagrama de bloques del microcontrolador.

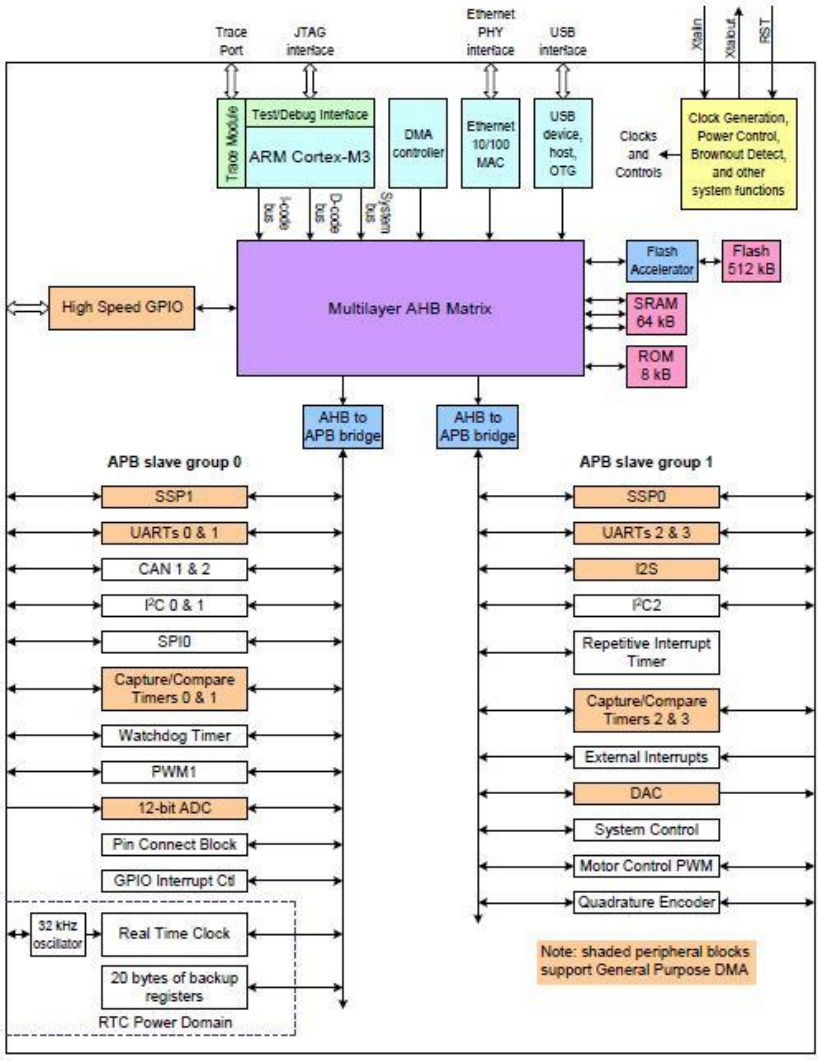


Ilustración 55: Diagrama de bloques del microcontrolador

Respecto a la interfaz USB este microcontrolador es compatible con la versión 2.0 y trabaja a full speed. Posee 16 endpoints (32 físicos ya que cada endpoint puede ser para transferencias OUT o IN). Estos endpoints además soportan los 4 tipos de transmisiones (bulk, control, isócronas o de datos).

La arquitectura del controlador del dispositivo USB se muestra a continuación:

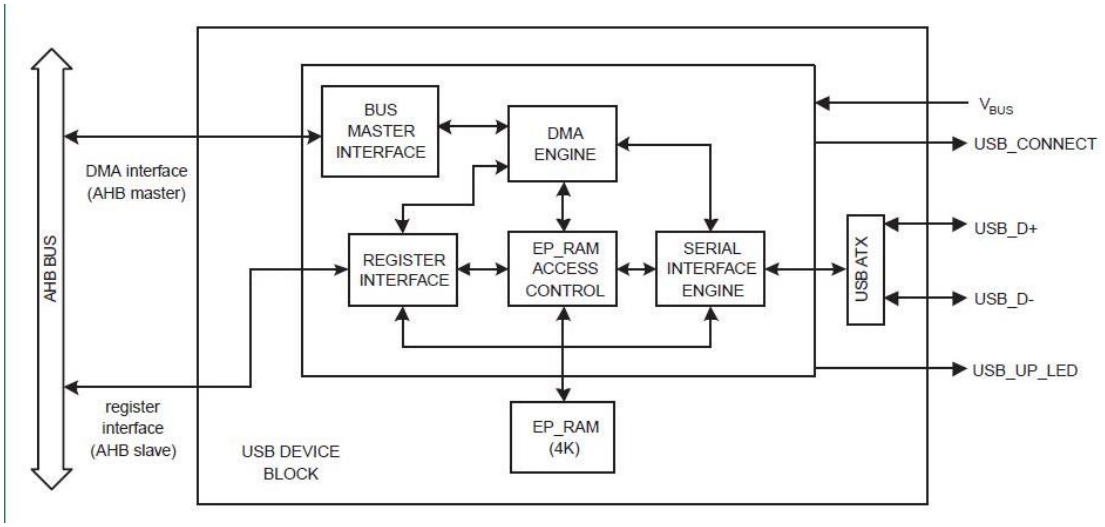


Ilustración 56: Arquitectura del controlador del dispositivo USB

Como se ve en la imagen anterior, hay diferentes interfaces entre el bus USB (USB_D+ y USB_D-) y el AHB BUS (el bus que conecta con la CPU del microcontrolador).

A continuación se va a explicar los diferentes bloques.

Transceiver analógico: Como se puede ver, el controlador del dispositivo USB tiene un transceiver analógico (USB ATX) que se encarga de convertir la información del bus a la comunicación serial.

Serial Interface Engine (SIE): La SIE tiene completamente implementada la capa del protocolo USB. El hardware está implementado para la velocidad necesaria y no necesita la implementación de ningún firmware. Se encarga de manejar la información de los buffer en los endpoint (EP_RAM) y el bus.

Las funciones de este bloque son las siguientes:

- Reconocer el patrón de sincronización.
- La conversión serial/paralelo.
- Del bit stuffing/de-stuffing.
- Generación y reconocimiento de las CRCs.
- Generación y reconocimiento de las PID.
- Reconocimiento de la address asignada.

- Generación y reconocimiento de los paquetes handshake.

Control de acceso a la EP_RAM : Este bloque se encarga de manejar la transferencia de datos de o desde la EP_RAM a las tres fuentes que pueden acceder a ella, que pueden ser la CPU a través del register interface, la SIE o el DMA engine.

ENDPOINT_RAM: Cada buffer de los endpoint está implementado en una SRAM basada en FIFO (first input first output).

Cada tarea que se realiza en los endpoints tiene reservado un espacio en las EP_RAM. Por otro lado, el espacio requerido depende del número del endpoint (el máximo tamaño de los paquetes de ese endpoint o si el endpoint soporta doble buffer).

DMA engine y Bus Master Interface: El acceso directo a la memoria (DMA) puede estar habilitado o no para un endpoint en concreto. Solo hay un DMA compartido para todos los endpoints. Si está habilitado el DMA engine transfiere datos entre la RAM en el AHB y las EP_RAM.

Cuando está transmitiendo datos el DMA funciona como un maestro en el bus AHB a través del Bus master interface.

Register interface: Este bloque permite a la CPU controlar las operaciones del controlador del dispositivo USB, también permite leer y escribir datos al controlador del dispositivo USB.

En la siguiente imagen se muestran todos los endpoints disponibles, el tipo de transmisión que soportan, la dirección, el tamaño máximo de sus paquetes y si poseen o no doble buffer.

Logical endpoint	Physical endpoint	Endpoint type	Direction	Packet size (bytes)	Double buffer
0	0	Control	Out	8, 16, 32, 64	No
0	1	Control	In	8, 16, 32, 64	No
1	2	Interrupt	Out	1 to 64	No
1	3	Interrupt	In	1 to 64	No
2	4	Bulk	Out	8, 16, 32, 64	Yes
2	5	Bulk	In	8, 16, 32, 64	Yes
3	6	Isochronous	Out	1 to 1023	Yes
3	7	Isochronous	In	1 to 1023	Yes
4	8	Interrupt	Out	1 to 64	No
4	9	Interrupt	In	1 to 64	No
5	10	Bulk	Out	8, 16, 32, 64	Yes
5	11	Bulk	In	8, 16, 32, 64	Yes
6	12	Isochronous	Out	1 to 1023	Yes
6	13	Isochronous	In	1 to 1023	Yes
7	14	Interrupt	Out	1 to 64	No
7	15	Interrupt	In	1 to 64	No
8	16	Bulk	Out	8, 16, 32, 64	Yes
8	17	Bulk	In	8, 16, 32, 64	Yes
9	18	Isochronous	Out	1 to 1023	Yes
9	19	Isochronous	In	1 to 1023	Yes
10	20	Interrupt	Out	1 to 64	No
10	21	Interrupt	In	1 to 64	No
11	22	Bulk	Out	8, 16, 32, 64	Yes
11	23	Bulk	In	8, 16, 32, 64	Yes
12	24	Isochronous	Out	1 to 1023	Yes
12	25	Isochronous	In	1 to 1023	Yes
13	26	Interrupt	Out	1 to 64	No
13	27	Interrupt	In	1 to 64	No
14	28	Bulk	Out	8, 16, 32, 64	Yes
14	29	Bulk	In	8, 16, 32, 64	Yes
15	30	Bulk	Out	8, 16, 32, 64	Yes
15	31	Bulk	In	8, 16, 32, 64	Yes

Ilustración 57: Configuración de los endpoints

Los registros son espacios de memoria directamente accesibles por la CPU con diferentes funcionalidades.

Por otro lado, los registros de la SIE son accesibles mediante unos comandos que se explicarán más adelante.

En la siguiente tabla se muestran todos los registros, menos los de DMA, en lo referido al controlador del dispositivo USB.

Name	Description	Access	Reset value	Address
Clock control registers				
USBClkCtrl	USB Clock Control	R/W	0	0x5000 CFF4
USBClkSt	USB Clock Status	RO	0	0x5000 CFF8
Device interrupt registers				
USBIntSt	USB Interrupt Status	R/W	0x8000 0000	0x400F C1C0
USBDevIntSt	USB Device Interrupt Status	RO	0x10	0x5000 C200
USBDevIntEn	USB Device Interrupt Enable	R/W	0	0x5000 C204
USBDevIntClr	USB Device Interrupt Clear	WO	0	0x5000 C208
USBDevIntSet	USB Device Interrupt Set	WO	0	0x5000 C20C
USBDevIntPri	USB Device Interrupt Priority	WO	0	0x5000 C22C
Endpoint interrupt registers				
USBEpIntSt	USB Endpoint Interrupt Status	RO	0	0x5000 C230
USBEpIntEn	USB Endpoint Interrupt Enable	R/W	0	0x5000 C234
USBEpIntClr	USB Endpoint Interrupt Clear	WO	0	0x5000 C238
USBEpIntSet	USB Endpoint Interrupt Set	WO	0	0x5000 C23C
USBEpIntPri	USB Endpoint Priority	WO	0	0x5000 C240
Endpoint realization registers				
USBReEp	USB Realize Endpoint	R/W	0x3	0x5000 C244
USBEpIn	USB Endpoint Index	WO	0	0x5000 C248
USBMaxPSize	USB MaxPacketSize	R/W	0x8	0x5000 C24C
USB transfer registers				
USBRxData	USB Receive Data	RO	0	0x5000 C218
USBRxPLen	USB Receive Packet Length	RO	0	0x5000 C220
USBTxData	USB Transmit Data	WO	0	0x5000 C21C
USBTxPLen	USB Transmit Packet Length	WO	0	0x5000 C224
USBCtrl	USB Control	R/W	0	0x5000 C228
SIE Command registers				
USBCmdCode	USB Command Code	WO	0	0x5000 C210
USBCmdData	USB Command Data	RO	0	0x5000 C214

Ilustración 58: Registros

A continuación se explicará algunos de ellos.

- **USBClkCtrl:** Este registro controla el reloj del controlador del dispositivo USB. Consta de 32 bits y para que el software pueda acceder a los demás registros los bits DEV_CLK_EN (bit 1) y AHB_CLK_EN (bit 4) tienen que estar puestos.

  Hay que decir que para que este registro esté funcional el bit PCUSB del registro PCONP (se encarga del control de los registro) tiene que estar puesto.

- **USBClkSt**: En este registro el software pone el bit correspondiente a los bits puestos en el registro anterior. Es solo de lectura; su utilidad consiste en poder devolver el estado del registro anterior.
- **USBIntSt:** Antes de hablar de este registro se va a hacer un pequeño inciso sobre las interrupciones.

    - Interrupciones: Todas las transmisiones OUT no-isócronas a los endpoints; es decir las transferencias de control, bulk o de interrupción, generan una interrupción cuando ha recibido un paquete sin ningún error. Por otro lado las transmisiones IN no-isócronas generan una interrupción cuando el paquete se ha enviado correctamente o cuando el paquete NAK es enviado (mediante la SIE).
      Para los endpoints isócronos las interrupciones son generadas cada 1 ms.

  El controlador USB tiene tres líneas de interrupción: de alta prioridad (bit USB_INT_REQ_HP), de baja prioridad (bit USB_INT_REQ_HL) o una interrupción del DMA(bit USB_INT_REQ_DMA). En este registro se indica el tipo de interrupción de que se trata. De esta manera el software puede determinar el estado de la interrupción simplemente leyendo este registro.

- **USBDevIntSt:** Este registro indica el estado de las interrupciones del dispositivo. Si hay una interrupción se indica con un 1 el tipo de interrupción de que se trate. En la siguiente imagen se puede observar las diferentes interrupciones que existen.

Bit	Symbol	Description
0	FRAME	The frame interrupt occurs every 1 ms. This is used in isochronous packet transfers.
1	EP_FAST	Fast endpoint interrupt. If an Endpoint Interrupt Priority register (USBEpIntPri) bit is set, the corresponding endpoint interrupt will be routed to this bit.
2	EP_SLOW	Slow endpoints interrupt. If an Endpoint Interrupt Priority Register (USBEpIntPri) bit is not set, the corresponding endpoint interrupt will be routed to this bit.
3	DEV_STAT	Set when USB Bus reset, USB suspend change or Connect change event occurs.
4	CCEMPTY	The command code register (USBCmdCode) is empty (New command can be written).
5	CDFULL	Command data register (USBCmdData) is full (Data can be read now).
6	RxENDPKT	The current packet in the endpoint buffer is transferred to the CPU.
7	TxENDPKT	The number of data bytes transferred to the endpoint buffer equals the number of bytes programmed in the TxPacket length register (USBTxPLen).
8	EP_RLZED	Endpoints realized. Set when Realize Endpoint register (USBReEp) or MaxPacketSize register (USBMaxPSize) is updated and the corresponding operation is completed.
9	ERR_INT	Error Interrupt. Any bus error interrupt from the USB device.

Ilustración 59: Registro USBDevIntSt

- **USBDevIntEn:** Escribiendo un 1 en este registro se habilita a que el bit correspondiente del registro anterior pueda ser puesto cuando ocurre una interrupción.

- **USBDevIntClr:** Escribir un "1" en este registro, elimina el correspondiente bit en el registro USBdevInSt. Un cero no tiene efecto.

- **USBDevIntSet:** Escribir un "1" en este registro, pone el correspondiente bit en el registro USBdeviInSt. Un cero no tiene efecto.

- **USBDevIntPri:** Este registro se utiliza para cambiar la prioridad de la interrupción, poniendo un "1" se habilita que la interrupción sea de alta prioridad como se muestra en la imagen. Los bits FRAME y EP_FAST no pueden estar puestos en "1" a la vez.

Bit	Symbol	Value	Description
0	FRAME	0	FRAME interrupt is routed to USB_INT_REQ_LP.
		1	FRAME interrupt is routed to USB_INT_REQ_HP.
1	EP_FAST	0	EP_FAST interrupt is routed to USB_INT_REQ_LP.
		1	EP_FAST interrupt is routed to USB_INT_REQ_HP.
31:2	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

Ilustración 60: Registro USBDevIntPri

- **USBEpIntSt:** Cada endpoint físico (32 endpoints) está representado por un bit en este registro indicando si ha generado una interrupción.

- **USBEpIntEn:** Al poner un "1" en este registro se habilita a que el bit correspondiente del registro anterior sea puesto cuando ocurre una interrupción en un endpoint.

- **USBEpIntClr:** Un 1 en este registro limpia el bit correspondiente al registro USBEpIntSt.

- **USBEpIntSet:** Escribir un 1 en este registro pone el bit correspondiente en el registro USBEpIntSt.

- **USBRxData:** En este registro es de donde la CPU lee el buffer del endpoint. Antes de que se lea el buffer los bits RED_EN y LOG_ENDPOINT del registro USBCtrl deben de estar debidamente puestos.

- **USBRxPLen:** Este registro contiene el número de bytes en el buffer del endpoint, que será leído mediante el registro anterior. Antes de que se lea el buffer los bits RED_EN y LOG_ENDPOINT del registro USBCtrl deben de estar debidamente puestos.

- **USBTxData:** Para transacciones IN, la CPU escribe los datos del endpoint en este registro. Antes de que se escriba en el buffer los bits WR_EN y LOG_ENDPOINT del registro USBCtrl deben de estar debidamente puestos.

- **USBTxPLen:** Este registro contiene el número de bytes que serán escritos en el endpoint. Antes de que se escriba en el buffer los bits WR_EN y LOG_ENDPOINT del registro USBCtrl deben de estar debidamente puestos

- **USBCtrl:** Este registro controla las operaciones de transferencia de datos. Se selecciona el buffer del endpoint correspondiente mediante LOG_ENDPOINT y habilita el poder leer o escribir mediante los bits RD_EN y WR_EN.

Bit	Symbol	Value	Description
0	RD_EN		Read mode control. Enables reading data from the OUT endpoint buffer for the endpoint specified in the LOG_ENDPOINT field using the USBRxData register. This bit is cleared by hardware when the last word of the current packet is read from USBRxData.
		0	Read mode is disabled.
		1	Read mode is enabled.
1	WR_EN		Write mode control. Enables writing data to the IN endpoint buffer for the endpoint specified in the LOG_ENDPOINT field using the USBTxData register. This bit is cleared by hardware when the number of bytes in USBTxLen have been sent.
		0	Write mode is disabled.
		1	Write mode is enabled.
5:2	LOG_ENDPOINT	-	Logical Endpoint number.
31:6	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

Ilustración 61: Registro USBCtrl

- **USBCmdCode:** Este registro es utilizado para enviar comandos y escribir en la SIE. Los comandos escritos aquí son enviados a la SIE y ejecutados ahí.

Bit	Symbol	Value	Description
7:0	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
15:8	CMD_PHASE		The command phase:
		0x01	Write
		0x02	Read
		0x05	Command
23:16	CMD_CODE/ CMD_WDATA		This is a multi-purpose field. When CMD_PHASE is Command or Read, this field contains the code for the command (CMD_CODE). When CMD_PHASE is Write, this field contains the command write data (CMD_WDATA).
31:24	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

Ilustración 62: Registro USBCmdCode

- **USBCmdData:** Este registro contiene los datos devueltos después de ejecutar un comando de la SIE.

A todas las funciones y los registros de la SIE se accede mediante comandos. Un acceso completo consiste en dos fases.

1. **Command phase:** En el campos CMD_PHASE del registro se pone el valor 0x05 (comando) y en el campo CMD_DCODE se pone el código del comando deseado.

2. **Data phase (optional):** Para escribir el campo CMD_PHASE del registro USBCmdCode, tiene que tener el valor 0x01(write) y en el campo CMD_WDATA estarán puestos los datos que se desean escribir. Para leer el campo CMD_PHASE del registro se pone el valor 0x02 (read) y en el campo CMD_DCODE se pone el código del comando deseado. Para leerlo se utiliza el registro USBCmdData.

En la siguiente imagen se muestra los comandos que se le pueden dar a la SIE

Command name	Recipient	Code (Hex)	Data phase
Device commands			
Set Address	Device	D0	Write 1 byte
Configure Device	Device	D8	Write 1 byte
Set Mode	Device	F3	Write 1 byte
Read Current Frame Number	Device	F5	Read 1 or 2 bytes
Read Test Register	Device	FD	Read 2 bytes
Set Device Status	Device	FE	Write 1 byte
Get Device Status	Device	FE	Read 1 byte
Get Error Code	Device	FF	Read 1 byte
Read Error Status	Device	FB	Read 1 byte
Endpoint Commands			
Select Endpoint	Endpoint 0	00	Read 1 byte (optional)
	Endpoint 1	01	Read 1 byte (optional)
	Endpoint xx	xx	Read 1 byte (optional)
Select Endpoint/Clear Interrupt	Endpoint 0	40	Read 1 byte
	Endpoint 1	41	Read 1 byte
	Endpoint xx	xx + 40	Read 1 byte
Set Endpoint Status	Endpoint 0	40	Write 1 byte
	Endpoint 1	41	Write 1 byte
	Endpoint xx	xx + 40	Write 1 byte
Clear Buffer	Selected Endpoint	F2	Read 1 byte (optional)
Validate Buffer	Selected Endpoint	FA	None

Ilustración 63: Comandos de la SIE

Para programar la MBED se han utilizado diferentes librerías a diferentes niveles como se muestra a continuación.

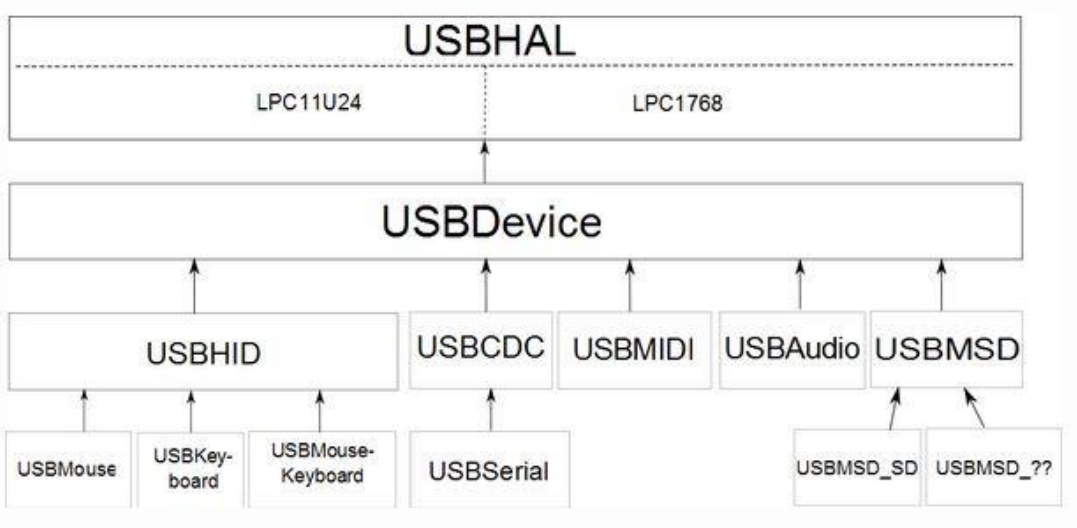


Ilustración 64: Librerías MBED USB

USB HAL: En esta capa están definidos todos los métodos de bajo nivel. En esta capa se implementa todo lo relacionado con los registros anteriormente relatados. En ella también están implementados mediante funciones todos los comandos referentes a la SIE. En esta librería se encuentra:

- El constructor para inicializar el controlador del dispositivo USB.
- La habilitación de las interrupciones de los endpoints.
- Definiciones de todos los registros referentes a la SIE.
- Funciones para añadir endpoints.
- Funciones para escribir o leer en los endpoints.

USB DEVICE: Esta librería es un paso intermedio entre la librería del tipo de dispositivo concreto que vayamos a utilizar y la librería USB HAL.

Por un lado en ella están implementadas las funciones para conectar y desconectar o escribir y leer en los endpoints aparte de añadirlos, estás funciones básicamente llaman a las funciones de la librería USB HAL.

También están implementadas las funciones necesarias, que son llamadas en las interrupciones del EP 0, para las request de setup del host (apartado 2.2.6) En ellas se configura la MBED como un dispositivo USB y se envía la información al host.

USB HID: Ésta es la librería que se va a utilizar, en ella están incluidas las dos anteriores y están implementadas funciones tanto para enviar o recibir datos, así como las funciones que son llamadas en las interrupciones del EP 0 para configurarlo como un dispositivo HID.

Igualmente, se utilizan otras librerías no representadas en la imagen anterior en las que solo se definen macros. Estas librerías son HID_Types.h, USBDescriptor.h, USBDevice_Types.h, USBEndpoints.h y USBEndpoints_LFP17_LCP23.

3 - PROGRAMAR LA MBED

3.1 - PRIMEROS PASOS

Antes de programar la MBED hay que instalar el firmware. El firmware es fácilmente descargable desde la página de MBED. Para instalarlo hay que conectar la MBED al ordenador mediante el conector de la MBED. Una vez reconocido por el ordenador se abre la carpeta correspondiente a la MBED que aparecerá como un disco extraíble más. Una vez abierto se copia el archivo dentro de esta carpeta y se reinicia la MBED. De esta manera el firmware ya estará instalado.

Con los programas se realiza un procedimiento parecido. Primero se escribe el programa en el compilador. El compilador es gratis y accesible online desde la página de MBED y trabaja utilizando el lenguaje C++. Una vez escrito el programa se compila, esto descargará un archivo .bin en el ordenador. Este archivo se copia en la carpeta de MBED y se resetea.

3.2 - PROGRAMA

Para comprobar la comunicación con la MBED se ha realizado un programa sencillo que consiste en apagar y encender un led cuando reciba la orden desde el PC y que envíe un “0” para indicar que el led está apagado o un “1” para indicar que está encendido. De esta manera se comprueba que pueden recibir y enviar datos sin errores.

El programa se muestra en el Anexo I.

A continuación se explica los procedimientos necesarios para la conexión de la MBED y su configuración como dispositivo USB, haciendo referencia a las funciones localizadas en las diferentes librerías que no se muestran en el programa.

Se incluye la librería MBED.h que proporciona las funciones básicas de la MBED y también la librería USBHID que incluye las librerías vistas en el apartado anterior.

Primero se llama a la función "USBHID" localizada en la librería USBHID en la que, como parámetro, se pone el tamaño en bytes de los datos que se van a enviar y recibir, en este caso es solo de 8 bytes. Esta función llama a la función "connect" localizada en la librería USBHAL. En esta función se habilitan las interrupciones del USB y mediante los comandos de la SIE "set device status" conecta el dispositivo. También se llama a la función "USBHAL". Esta función habilita el controlador del dispositivo USB, los registros del reloj (ver apartado 2.3.3), el tamaño máximo de los EP0 (64 bits) y habilita los pines 29 y 30 como D+ y D-.

En este momento el dispositivo ya está encendido y preparado para recibir las peticiones del host descritas en la sección 2.2.6. En este momento el host empieza a enviar sus peticiones, tales como que le dé sus descriptores o asignarle una address. Esto se hace mediante funciones alojadas en la librería USBDevice. Estas funciones son llamadas durante las interrupciones.

Primero se llama a la función "EP0setupCallback" y ésta a su vez llama a la función "controlsetup" localizada en USBDevice, ésta a su vez llama a la función "decodesetuppacket" en donde los datos recibidos son ordenados en una estructura siguiendo las estructuras de los paquetes setup es decir con un campo para brequestype, brequest, wvalue, etc. (apartado 2.2.6). Después de decodificar el paquete, se llama a la función virtual "USBCallback_request" definida en la librería USBHID que analiza las peticiones del host, en el caso de que la petición sea obtener el descriptor de la interfaz, éste envía su descriptor (HID Descriptor).

En el caso de que fuese otra petición se llama a la función "requestsetup" localizada en la librería USBDevice. Esta función analiza el paquete setup previamente decodificado y dependiendo de qué tipo de petición sea lo envía a una función o a otra; hay una función para cada petición del host en USBDevice. Hay que decir que si la petición es la de obtener el descriptor (get descriptor), se llama a la función pertinente en la librería USBDevice. En el caso de que pida el device descriptor, esta librería lo tiene implementado mediante la función "devicedesc", pero si por el contrario pide el configuration descriptor, se llama a la función "confdes" localizada en USBHID donde están implementadas las descripciones necesarias.

Cuando se recibe la petición del host set address, se llama a la función pertinente en la librería USBDevice. Esta función a su vez llama a la función "setaddress" localizada en la librería USBHAL. En esta función se le proporciona los comandos a la SIE para que asigne la dirección que el host ha enviado.

También se llama a la función "setconfiguration" localizada en la librería USBHID que añade el endpoint 1 definido en la librería USBEndpoints como un endpoint de interrupción (por lo tanto la comunicación será a través de transmisiones de interrupción). La función llama a la función "enableendpointevent", localizada en USBHAL, que pone el correspondiente bit del registro USBEpIntEn.

Una vez que el host ya le ha asignado la address mediante la petición set address y tiene todos los descriptores del dispositivo, éste ya está configurado y preparado para funcionar.

Por lo genera,l todas las funciones llaman a funciones localizadas en la librería USBHAl, que se encarga, mediante los comando de la SIE, de escribir o leer en los endpoints, habilitar interrupciones en los diferentes endpoints o habilitar y deshabilitar los endpoints. La manera de escribir y leer en los endpoints se detalla más adelante. También se encarga de escribir en los registros.

Siguiendo con el programa, a continuación se asigna la variable led al LED1 de la MBED, por la tanto, cuando la variable led tenga el valor 1 se encenderá y cuando tenga el valor 0 se apagará.

Con el siguiente comando "if(hid.readNB(&recv_report))", si se detecta que se han recibido datos, la MBED hace lo que tiene dentro del if y guarda el valor leído en la variable recv_report, cuando no se lee nada, la MBED no hace nada. La función "readNB" está alojada en la librería HID. Esta función llama a una función alojada en USBDevice en el que le indica qué endpoint debe de leer. Esta función comprueba que el dispositivo está configurado y si es así llama a la función "endpointreadcore" de la librería USBHAL. En esta función si la transmisión de datos está en proceso, espera a que se termine, y por último se llama a la función "enpointreadcore" localizada en esta misma librería.

En esta función se ponen los bits oportunos al registro USBCtrl (apartado 2.3.3), comprueba el tamaño de los datos mediante el registro USBRxLen y por último guarda en una variable el valor del registro USBRxData. Para finalizar, limpia el registro USBCtrl y utiliza el comando de la SIE clearbuffer para borrar los datos del buffer del endpoint.

Al recibir un mensaje y haber guardado los datos en la variable recv_report, se comprueba el valor de dicha variable, para apagar o encender el led. Por último se envía un 1 o un 0 indicando que ha recibido el mensaje y que ha apagado o encendido el led. Para enviar un mensaje se utiliza la funcion "send", que utiliza un proceso similar a la función de leer, pero en lugar de leer del registro USBRxData, escribe en el registro USBTxData.

3.3 – COMUNICACIÓN CON EL PC

Para la comunicación con el ordenador se ha decidido utilizar Python como interface utilizando el módulo pywinUSB.

El programa utilizado se muestra en el Anexo II.

En este programa se definen tres funciones. La función "raw_test", que busca todos los dispositivos USB conectados, escribe sus características y los enumera y pide que se elija uno de ellos. Esta función devuelve el dispositivo elegido. A continuación se define la función "hacer", en esta función se llama a la función "device.set_raw_data_handler" y como argumento se pone la función "recibido" de la que se tratará más adelante. Con la función "device.set_raw_data_handler" se llama a la función "recibido" cada vez que se reciba un mensaje. Siguiendo con la función "hacer", ésta da la opción de encender o apagar el led del MBED o bien salir si se quiere volver otra vez a la función "raw_test". Al escribir la opción deseada se envía un mensaje a las MBED con el dato escrito. Por otro lado, se comprueba la opción elegida y si es un 3 se sale de la función "hacer" y da la opción de salir del programa o de volver a la función "raw_test". Por el contrario, si la opción elegida es un 1 o un 0, envía los datos y vuelve a dar la opción de encender o apagar el led o salir. Por último, respecto a la función "recibido", en su argumento se encuentran los

datos recibidos de la MBED, en esta función se comparan y si es un 1 se indica que el led está encendido y si es un 0 que está apagado.

4 – CONCLUSIONES Y TRABAJOS FUTUROS.

En los apartados anteriores se ha explicado el procedimiento para poder establecer comunicación desde el ordenador con una plataforma MBED de tal manera que sea capaz de poder enviar y recibir datos; esto se ha comprobado y funciona correctamente.

En la siguiente imagen se muestra la pantalla de comandos de Python, en la que primero se observan todos los dispositivos USB HID conectados al sistema y sus características. Una vez elegido el dispositivo, en este caso MBED, se ofrecen tres opciones, "apagar", "encender" o "cambiar de dispositivo". En la pantalla se observa que al enviar un "1" (encender) desde el ordenador, se enciende el led y la MBED, a su vez, envía igualmente un "1" al ordenador indicando que se ha encendido y reflejando en la pantalla "encendido". Al apagar se verifica el mismo proceso, se envía un "0" (apagar) desde el ordenador que apaga el led, y seguidamente por la MBED se envía un "0" al ordenador y se indica en la pantalla "apagado". Se constata, por tanto, que funciona perfectamente la comunicación entre el ordenador y la MBED.

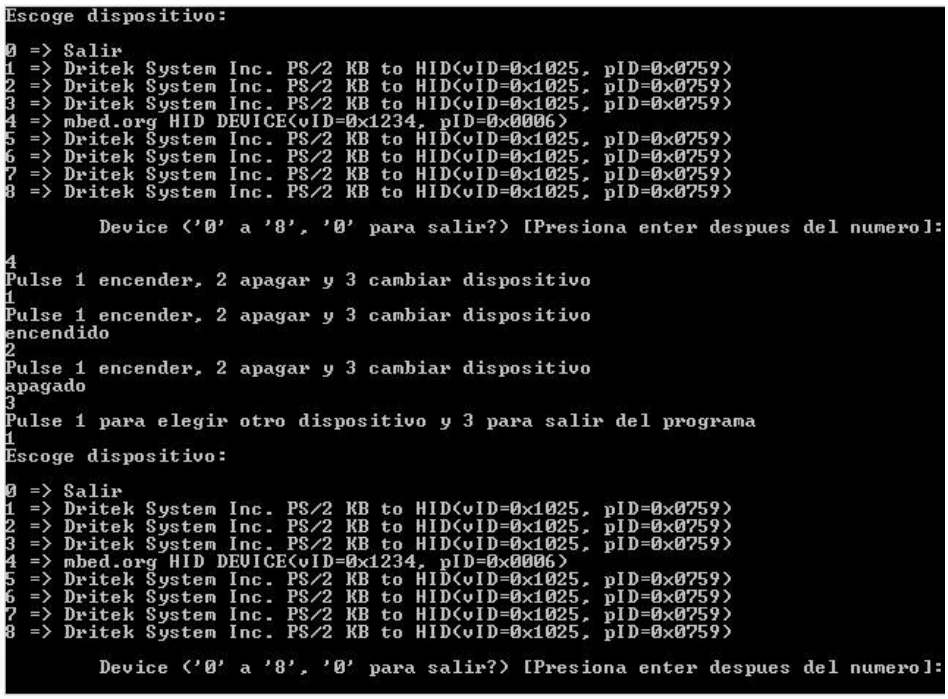


Ilustración 65: Programa de Python

En el robot TEO es necesario poder establecer comunicación con varias MBED a la vez. En un sistema USB convencional se pondría un Hub y se tiraría un cable para cada MBED, pero debido a la arquitectura hardware del propio robot, ya construida, no es posible tirar un cable para cada MBED, solo admite utilizar un solo bus para todas ellas.

Se intentó, en un primer momento, conectar todas las MBED a la vez al mismo bus, con distinto ID cada una de ellas, comprobándose que no daba los resultados esperados, ya que la comunicación no era posible, debido a que el host, al estar los dispositivos conectados al mismo puerto, no los reconocía, dado el diseño del USB, tal y como ha sido explicado.

Con posterioridad, se optó por conectar todas las MBED con el mismo ID para que el host las reconociese como un único dispositivo. Para reconocer las diversas MBEDs y discriminar con la que se quiere efectuar la comunicación, se le asignó un identificador a cada MBED localizado en el cuerpo del mensaje, de tal forma que los mensajes contenían dos identificadores, uno de ellos propio del host, que era el mismo para todas las plataformas y el otro, asignado de forma individual a cada MBED, que las diferenciaba, así las MBEDs comprueban el mensaje, y si se corresponde con su identificador, actúan consecuentemente. Al comprobar los resultados se concluyó que la opción no era factible. Si solo se pretendía enviar información desde el host a la MBED, el resultado era positivo, las MBEDs recibían perfectamente el mensaje del host. El problema se presentaba cuando los datos eran enviados desde la MBED al host, esto es debido a que el host enviaba el paquete de petición de datos (token) con la address de todas las MBED, y las SIEs de las MBEDs, al recibir este mensaje enviaban automáticamente la información almacenada en el buffer del endpoint, y como los datos eran diferentes, colisionaban produciendo un error en la comunicación.

Finalmente, no se pudo alcanzar el objetivo pretendido, pese a haber ensayado diferentes opciones.

Como trabajos futuros se plantea por un lado, una modificación del software y por otro, la modificación del hardware para establecer la comunicación pretendida.

- **Modificación del software**: Esta es la mejor opción; en cierto modo es bastante parecido al sistema OTG que se está empezando a implementar en otros dispositivos, como por ejemplo los móviles. Consiste en que un dispositivo pueda funcionar como esclavo (cuando se conecta el móvil al ordenador) o como maestro (conectar un pendrive directamente al móvil).

  Consistiría en lo siguiente:

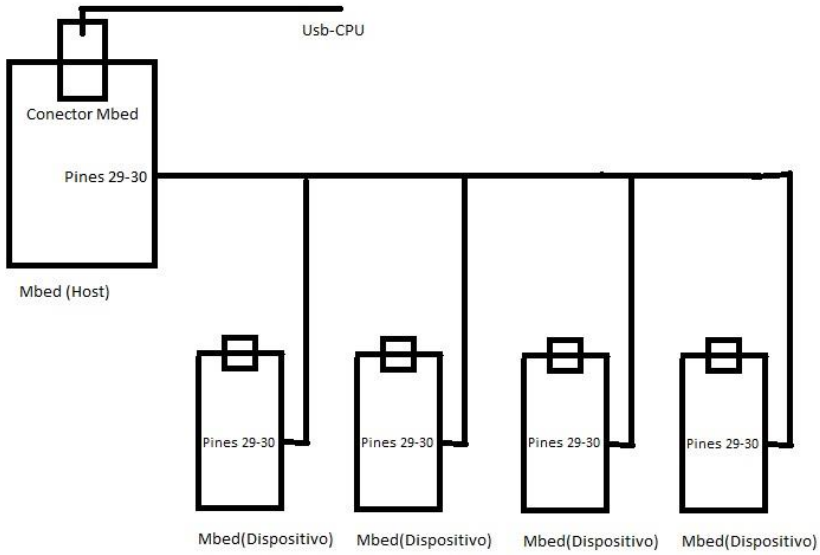


Ilustración 66: Esquema modificación software

Como se puede apreciar, consiste en utilizar una MBED como host y como esclavo a la vez. Todas las demás MBED enviarán la información a esta MBED mediantes los pines 29 y 30 (D+ y D-) y ésta la enviará a la CPU de TEO mediante el conector de la MBED.

El problema de este método es la falta de Hubs, sería necesario a la hora de implementar el host en la MBED controladora, hacerlo de tal manera que no necesite los Hubs. Debido a que en la arquitectura USB los Hubs son una pieza esencial, si se llegara a conseguir, esta opción podría no llegar a cumplir los requerimientos mínimos en cuanto a los errores en la comunicación y velocidad de transferencia.

- **Modificación del hardware**: Si la anterior solución no fuera factible, como última opción queda una modificación del hardware.

  Esta modificación consistiría en colocar un controlador Hubs, como por ejemplo el de microchip "USB 2421", que es un circuito integrado con un upstream y dos downstream.

  Se utilizaría al igual que el caso anterior una MBED como host y se colocaría un Hub por cada MBED de la siguiente manera.

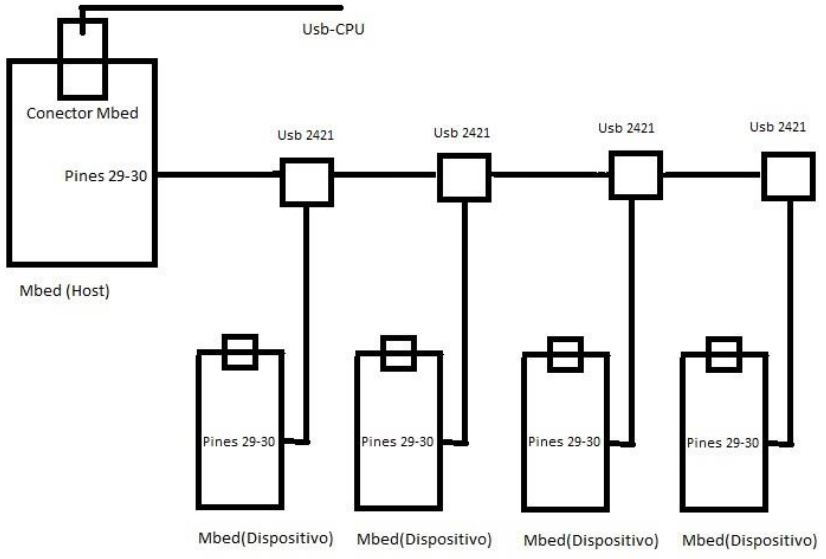


Ilustración 67: Esquema modificación hardware

El problema de esta solución radica precisamente en el hardware. El estar ya diseñado y en parte construido, con la idea de no incorporar los Hubs a las comunicaciones, supone un problema a la hora de querer añadir un IC adicional. Como las placas donde van a estar ubicadas las MBED ya están construidas, una opción sería colocar estos controladores en los conectores de los cables.

5 - BIBLIOGRAFÍA

Páginas web:

[1] http://www.noticias.tudiscovery.com/cinco-impresionantes-robots-humanoides-reemplazando-roles-de-personas/

[2] https://www.mbed.org/

[3] http://www.usbmadesimple.co.uk

[4] http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/alvarez_v_ce/apendiceA.pdf

[5] http://www.subdivx.com/

[6] http://www.conclase.net/

[7] http://www.wikipedia.com/

[8] http://www.beyondlogic.org/usbnutshell/

[9] http://www.microchip.com/

[10] http://www.python.org/

[11] https://github.com/rene-aguirre/pywinusb/

[12] http://portal.uc3m.es/portal/page/portal/actualidad_cientifica/noticias/conferencia_yokoi

Proyectos:

[13] Análisis de la arquitectura hardware modular y descentralizada del robot humanoide TEO; autor Juan Miguel García Haro, Universidad Carlos III de Madrid; junio 2014.

[14] Diseño del sistema de conexionado electrónico para los brazos del robot humanoide RH2; autor Eduardo Martínez Fernández, Universidad Carlos III de Madrid; junio 2011.

[15] Diseño e implementación Encoder Absoluto CANopen; autor Julián Fortes Monteiro; Universidad Carlos III de Madrid; julio 2009.

[16] Estado del Arte; Autor Andrés Aguirre, Pablo Fernández y Carlos Grossy, Universidad de la República Oriental de Uruguay; 14 de diciembre de 2007.

Otros:

[17] C++, Manual Teórico Práctico; Autor Alan D. Osorio Rojas; noviembre de 2006.

[18] Universal Serial Bus Specification, Revision 2.0; Compaq, Hewlett-Packard; Intel, Lucent, Microsoft, Nec, Philips; 27 de abril de 2000.

[19] User Manual, UMD 10360, LCP 176x/5x, Revisión 3.1; 2 de a bril de 2014.

6 – ANEXOS

6.1- Anexo I. Programa MBED

```
#include "MBED.h"
#include "USBHID.h"
USBHID hid(1, 1); // Declaramos el dispositivo HID
DigitalOut led(LED1);//Habilita el LED1 y asigna variable
HID_REPORT send_report;
HID_REPORT recv_report;
 int main(void) {
   while (1) {
      send_report.length= 1;
       if(hid.readNB(&recv_report)) {//leo el mensaje
        if (recv_report.data[0]==1){//comparo valor
           led=1; // enciendo el led
           send_report.data[0]=1;
           hid.send(&send_report);//envío estado del led
          }
        if (recv_report.data[0]==2){//leo el mensaje
          led=0;//apago el led
          send_report.data[0]=0;
          hid.send(&send_report); );//envío estado del led
         }
      }
    }
}
```

6.2 - Anexo II. Programa Python

```
from time import sleep
from msvcrt import kbhit
import pywinUSB.hid as hid
def raw_test():
    tt=1
    all_hids = hid.find_all_hid_devices()
    if all_hids:
        while tt==1:
            print("Escoge el dispositivo:\n")
            print("0 => Salir")
            for index, device in enumerate(all_hids):
                device_name = unicode("{0.vendor_name} {0.product_name}" \
                      "(vID=0x{1:04x}, pID=0x{2:04x})"\
                      "".format(device, device.vendor_id, device.product_id))
                print("{0} => {1}".format(index+1, device_name))
            print("\n\tDevice ('0' a '%d', '0' para salir?) " \
                  "[Presiona enter después del número]:" % len(all_hids))
            index_option = raw_input()
            int_option = int(index_option)
            if int_option:
                device = all_hids[int_option-1]
                tt=0
    else:
        print("No hay dispositivos HID disponibles")
    return device

def hacer():
 to=1
 device.open()
 device.set_raw_data_handler(recibido)
 while to!=3:
  print " Pulse 1 encender, 2 apagar y 3 cambiar dispositivo"
  acion= raw_input()
```

```
  to= int(acion)
  report=device.find_output_reports()[0]
  report[MBED_usage]=to
  report.send()
def recibido(data):
   if data[1]==0:
    print "apagado"
   if data [1]==1:
    print "encendido"
if __name__ == '__main__':
    # first be kind with local encodings
    import sys
    if sys.version_info >= (3,):
        # as is, don't handle unicodes
        unicode = str
        raw_input = input
    else:
            import codecs
        sys.stdout = codecs.getwriter('mbcs')(sys.stdout)
a=1
MBED_usage=hid.get_full_usage_id(0xffab, 0x02)
while a!=3:
device=raw_test()
hacer()
 print "Pulse 1 para elegir otro dispositivo y 3 para salir del programa"
a= int(raw_input())
```

6.3 – Anexo III. Microcontrolador HUB USB2412 datasheet

2-Port USB 2.0 Hi-Speed Hub Controller

PRODUCT FEATURES Datasheet

General Description

The SMSC USB2412 hub is a low-power, single transaction translator (STT) hub controller IC with two downstream ports for embedded USB applications. The SMSC hub controller supports low-speed, full-speed, and hi-speed (if operating as a hi-speed hub) downstream devices on all of the enabled downstream ports.

Features

- Fully integrated USB termination and pull-up/pull-down resistors
- Supports a single external 3.3 V supply source; internal regulators provide 1.2 V internal core voltage
- On-chip 24 MHz crystal driver or external 24 MHz clock input
- ESD protection up to 4 kilovolts on all USB pins
- Supports self-powered operation
- Contains a built-in default configuration; no external configuration options or components are required
- Downstream ports as optional non-removable ports
- Supports compound devices on a port-by-port basis
- 28-pin QFN (5 x 5 mm) lead-free RoHS compliant package
- Supports the commercial temperature range: 0°C to +70°C

Highlights

- High performance, low-power, small footprint hub controller IC with two downstream ports
- Fully compliant with the *USB 2.0 Specification* 1.
- 28QFN low pin count package
- Optimized for minimal bill-of-materials and low cost designs

Applications

- Automobile/home audio systems
- Cable/DSL modems
- Embedded systems
- Gaming consoles
- HDD enclosures
- IP telephony
- KVM switches
- LCD monitors and TVs
- Multi-function USB peripherals
- Mobile PC docking
- PC motherboards
- PC media drive bay
- Portable hub boxes
- Point-of-Sale (POS) systems
- Printers and scanners
- Server front panels
- Set-top boxes, DVD players, DVR/PVR
- Thin client terminals

SMSC USB2412 **DATASHEET** Revision 1.3 (12-27-11)

2-Port USB 2.0 Hi-Speed Hub Controller

Datasheet

Order Numbers:

ORDER NUMBERS*	LEAD-FREE ROHS COMPLIANT PACKAGE	PACKAGE SIZE	REEL SIZE
USB2412-DZK	28-Pin QFN Lead-Free, RoHS Compliant Package (includes tape and reel option)	5 x 5 x 0.5 mm	-
USB2412-DZK-TR			

This product meets the halogen maximum concentration values per IEC61249-2-21

For RoHS compliance and environmental information, please visit www.smsc.com/rohs

80 ARKAY DRIVE, HAUPPAUGE, NY 11788 (631) 435-6000 or 1 (800) 443-SEMI

Conventions

Within this manual, the following abbreviations and symbols are used to improve readability.

Example	Description
BIT	Name of a single bit within a field
FIELD.BIT	Name of a single bit (BIT) in FIELD
x...y	Range from x to y, inclusive
BITS[m:n]	Groups of bits from m to n, inclusive
PIN	Pin Name
zzzzb	Binary number (value zzzz)
0xzzz	Hexadecimal number (value zzz)
zzh	Hexadecimal number (value zz)
rsvd	Reserved memory location. Must write 0, read value indeterminate
code	Instruction code, or API function or parameter
Section Name	Section or Document name
x	Don't care
<Parameter>	<> indicate a Parameter is optional or is only used under some conditions
{,Parameter}	Braces indicate Parameter(s) that repeat one or more times
[Parameter]	Brackets indicate a nested Parameter. This Parameter is not real and actually decodes into one or more real parameters.

Chapter 1 Block Diagram

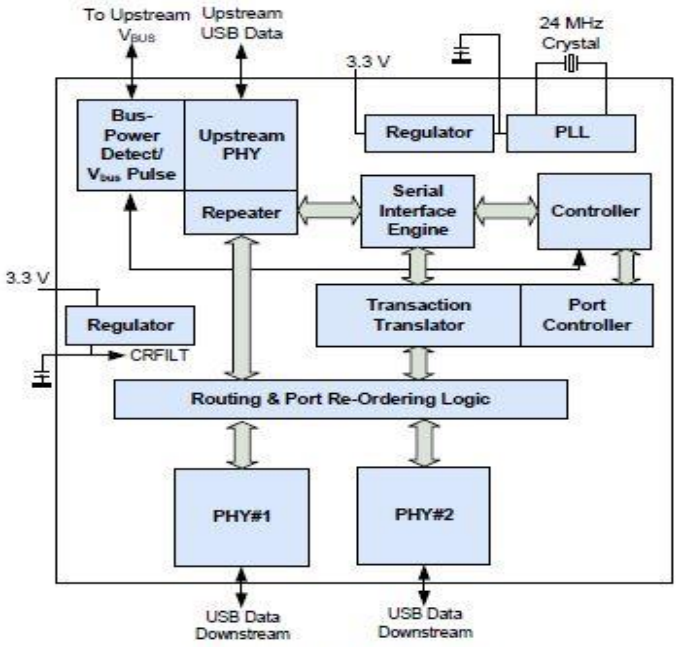


Figure 1.1 USB2412 Block Diagram

Chapter 2 Pin Descriptions

This chapter is organized by a set of pin configurations followed by a corresponding pin list organized by function according to their associated interface. A detailed description list of each signal (named in the pin list) is organized by function in Table 2.2, "USB2412 Pin Descriptions," on page 10. Refer to Table 2.3, "Buffer Type Descriptions," on page 12 for a list of buffer types.

The "N" symbol in the signal name indicates that the active, or asserted, state occurs when the signal is at a low voltage level. When "N" is not present after the signal name, the signal is asserted when it is at the high voltage level.

The terms assertion and negation are used exclusively. This is done to avoid confusion when working with a mixture of "active low" and "active high" signals. The term assert, or assertion, indicates that a signal is active, independent of whether that level is represented by a high or low voltage. The term negate, or negation, indicates that a signal is inactive.

2.1 Pin Configuration

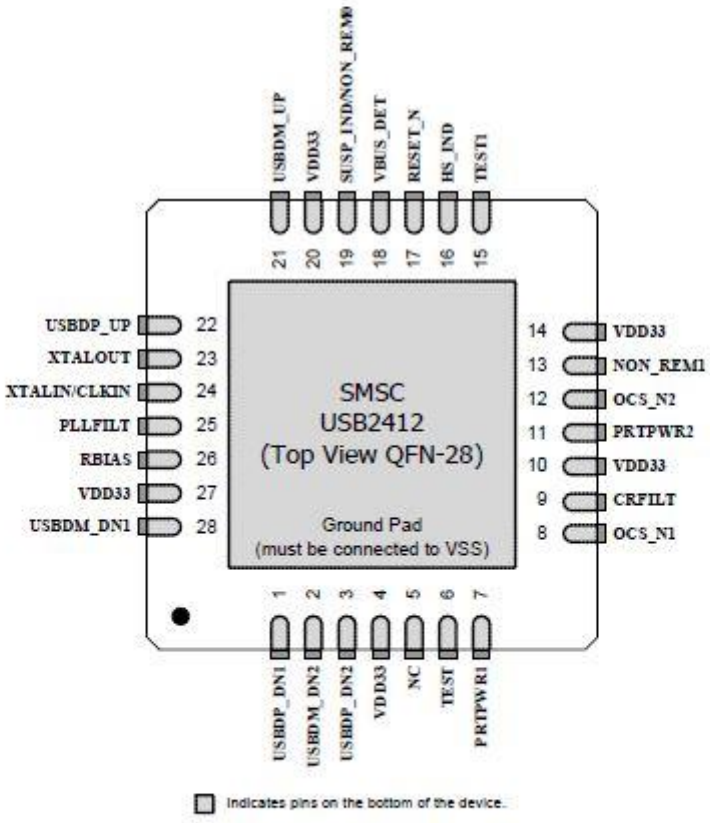


Figure 2.1 USB2412 28-Pin QFN

2.2 28-Pin Table

Table 2.1 USB2412 28-Pin Table

UPSTREAM USB 2.0 INTERFACES (3 PINS)			
USBDP_UP	USBDM_UP	VBUS_DET	
DOWNSTREAM 2-PORT USB 2.0 INTERFACES (9 PINS)			
USBDP_DN1	USBDM_DN1	USBDP_DN2	USBDM_DN2
PRT_PWR1	PRT_PWR2	OCS_N1	OCS_N2
RBIAS			
MISC (7 PINS)			
RESET_N	TEST	XTALIN / CLKIN	XTALOUT
NON_REM1	SUSP_IND / NON_REM0	HS_IND	
POWER, GROUND, AND NO CONNECTS (9 PINS)			
(5) VDD33	CRFILT	PLLFILT	VSS
NC			
TOTAL 28			

2.3 Pin Descriptions (Grouped by Function)

Table 2.2 USB2412 Pin Descriptions

PIN #	SYMBOL	BUFFER TYPE	DESCRIPTION
			UPSTREAM USB 2.0 INTERFACES
21 22	USBDM_UP USBDP_UP	IO-U	USB Bus Data: connect to the upstream USB bus data signals (host, port, or upstream hub).
18	VBUS_DET	I/O12	Detect Upstream VBUS Power: detects the state of upstream VBUS power. The SMSC hub monitors **VBUS_DET** to determine when to assert the internal D+ pull-up resistor which signals a connect event. When designing a detachable hub, this pin should be connected to VBUS on the upstream port via a 2 to 1 voltage divider. For self-powered applications with a permanently attached host, this pin must be connected to a dedicated host control output, or connected to 3.3 V domain that powers the host.' According to Section 7.2.1 of the *USB 2.0 Specification* 1., a downstream port can never provide power to its D+ or D- pull-up resistors unless the upstream port's VBUS is in the asserted (powered) state. **VBUS_DET** monitors the state of the upstream VBUS signal and will not pull-up the D+ resistor if VBUS is not active. If VBUS goes from an active to an inactive state (Not Powered), the hub will remove power from the D+ pull-up resistor within 10 seconds.
			DOWNSTREAM USB 2.0 INTERFACES
1 3 28 2	USBDP_DN1 USBDP_DN2 USBDM_DN1 USBDM_DN2	IO-U	Hi-Speed USB Data: connect to the downstream USB peripheral devices attached to the hub's ports.
7 11	PRTPWR1 PRTPWR2	I/O12	USB Power Enable: enables power to USB peripheral devices that are downstream, where the hub supports active high power controllers only.
8 12	OCS_N1 OCS_N2	IPU	Over-Current Sense: input from external current monitor indicating an over-current condition. This pin contains an internal pull-up to the 3.3 V supply.
26	RBIAS	I-R	USB Transceiver Bias: a 12.0 kΩ (+/- 1%) resistor is attached from ground to this pin to set the transceiver's internal bias settings.
			MISC
13	NON_REM1	I/O	Non-removable Port Strap Option: this pin is sampled (in conjunction with **SUSP_IND/NON_REM0**) at **RESET_N** negation to determine if ports [2:1] contain permanently attached (non-removable) devices: NON_REM[1:0] = 00: all ports are removable NON_REM[1:0] = 01: port 1 is non-removable NON_REM[1:0] = 10 and 11: ports 1 and 2 are non-removable See Section 2.5, "Strap Option Pins" for details.

Table 2.2 USB2412 Pin Descriptions (continued)

PIN #	SYMBOL	BUFFER TYPE	DESCRIPTION
19	SUSP_IND /	I/O	Active/Suspend Status LED (suspend indicator): indicates the USB hub state. See Section 2.5, "Strap Option Pins" for details. NON_REM0 = 0: **SUSP_IND** is active high NON_REM0 = 1: **SUSP_IND** is active low negated = unconfigured, or configured and in USB suspend asserted = hub is configured and is active (i.e., not in suspend)
	NON_REM0		Non-removable Port Strap Option: this pin is sampled (in conjunction with **NON_REM1**) at **RESET_N** negation to determine if ports [2:1] contain permanently attached (non-removable) devices: NON_REM[1:0] = 00: all ports are removable NON_REM[1:0] = 01: port 1 is non-removable NON_REM[1:0] = 10 and 11: ports 1 and 2 are non-removable See Section 2.5, "Strap Option Pins" for details.
16	HS_IND	I/O12	Hi-Speed Upstream Port Indicator **Note:** An LED can be attached for visual indication. See Section 2.5, "Strap Option Pins" for details. When an LED is not used, this pin requires a 50 kΩ or higher resistor to ground. negated = the hub is connected at FS asserted = the hub is connected at HS
24	XTALIN	ICLKx	24 MHz Crystal Input: this pin connects to either one terminal of the crystal or to an external 24 MHz clock when a crystal is not used.
	CLKIN		External Clock Input: this pin connects to either one terminal of the crystal or to an external 24 MHz clock when a crystal is not used.
23	XTALOUT	OCLKx	24 MHz Crystal Output: this is the other terminal of the crystal circuit with 1.2 V p-p output and a weak (< 1mA) driving strength. When an external clock source is used to drive **XTALIN/CLKIN**, leave this pin unconnected, or use with appropriate caution.
6	TEST	IPD	Treat as a no connect pin or connect to ground. No trace or signal should be routed or attached to this pin.
17	RESET_N	IS	RESET Input: the system can reset the chip by driving this input low, where the minimum active low pulse is 1 µs.
			POWER, GROUND, and NO CONNECTS
9	CRFILT		VDD Core Regulator Filter Capacitor: this pin can have up to 0.1 µF low-ESR capacitor to VSS, or be left unconnected.
4 10 14 20 27	VDD33		3.3 V Power
25	PLLFILT		PLL Regulator Filter Capacitor: this pin can have up to 0.1 µF low-ESR capacitor to VSS, or be left unconnected.
15	TEST1		This pin must be connected to VSS.

Table 2.2 USB2412 Pin Descriptions (continued)

PIN #	SYMBOL	BUFFER TYPE	DESCRIPTION
29	ePAD		Ground Pad/ePad: this pin must be connected to VSS for the device and must be tied to ground with multiple vias
5	NC		No Connect: no signal or trace should be routed or attached to these pins.

2.4 Buffer Type Descriptions

Table 2.3 Buffer Type Descriptions

BUFFER	DESCRIPTION
I/O	Input/Output
IPD	Input with internal weak pull-down resistor
IPU	Input with internal weak pull-up resistor
IS	Input with Schmitt trigger
I/O12	Input/Output buffer with 12 mA sink and 12 mA source
ICLKx	XTAL clock input
OCLKx	XTAL clock output
I-R	RBIAS
I/O-U	Analog Input/Output defined in USB specification

Chapter 6 Package Outline

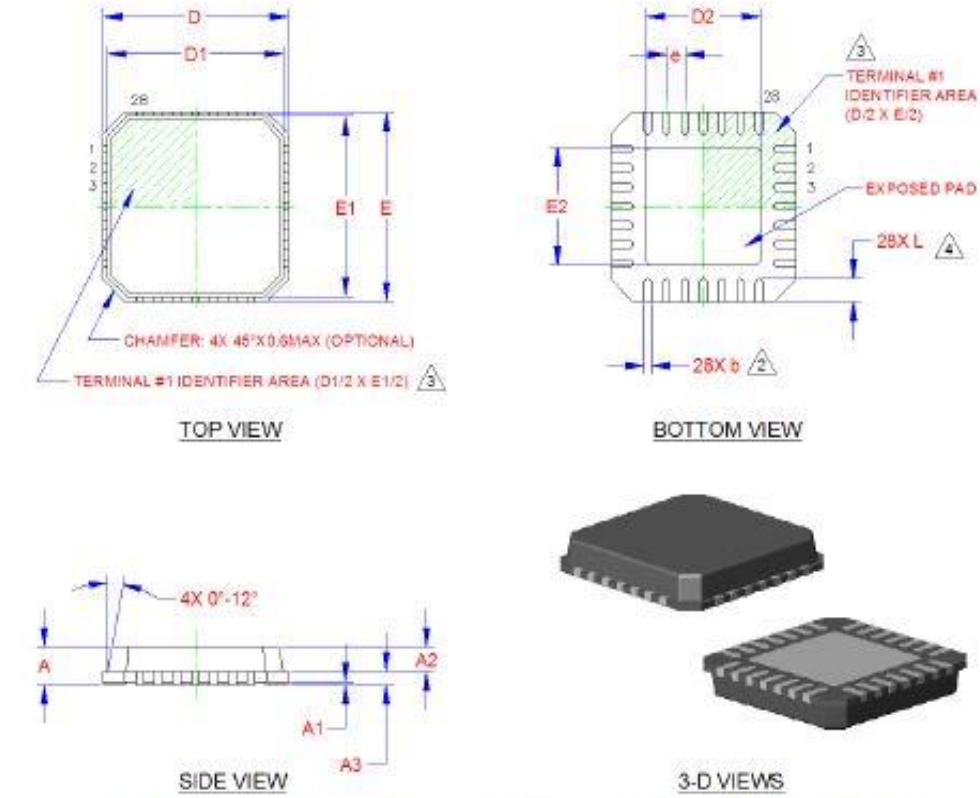


Figure 6.1 USB2412 28-Pin QFN Package Outline (5x5 mm Body, 0.5 Pitch, 3.1 ePad)

Table 6.1 Package Parameters

	MIN	NOMINAL	MAX	NOTE	REMARKS
A	0.80	0.85	1.00	-	Overall Package Height
A1	0	0.02	0.05	-	Standoff
A2	0.60	-	0.80	-	Mold Cap Thickness
D/E	4.90	5.00	5.10	-	X/Y Overall Body Size
D1/E1	4.55	4.75	4.95	-	X/Y Mold Cap Size
D2/E2	3.00	3.10	3.20	-	X/Y Exposed Pad Size
L	0.30	0.40	0.50	-	Terminal Length
b	0.18	0.25	0.30	2	Terminal Width
K	0.45	0.55	-	-	Terminal to ePad Clearance
e		0.50 BSC		-	Terminal Pitch

Notes:

1. All dimensions are in millimeters.
2. Position tolerance of each terminal and exposed pad is ±0.05 mm at maximum material condition. Instances of dimension "b" apply to plated terminals and is measured between 0.15 and 0.33 mm from the terminal tip.
3. Details of terminal #1 identifier are optional. However, they must be located within the area indicated.
4. Coplanarity zone applies to exposed pad and terminals.

6.4 – Anexo IV. Microcontrolador LCP 1768

LPC1769/68/67/66/65/64/63

32-bit ARM Cortex-M3 microcontroller; up to 512 kB flash and 64 kB SRAM with Ethernet, USB 2.0 Host/Device/OTG, CAN

Rev. 9.5 — 24 June 2014 Product data sheet

1. General description

The LPC1769/68/67/66/65/64/63 are ARM Cortex-M3 based microcontrollers for embedded applications featuring a high level of integration and low power consumption. The ARM Cortex-M3 is a next generation core that offers system enhancements such as enhanced debug features and a higher level of support block integration.

The LPC1768/67/66/65/64/63 operate at CPU frequencies of up to 100 MHz. The LPC1769 operates at CPU frequencies of up to 120 MHz. The ARM Cortex-M3 CPU incorporates a 3-stage pipeline and uses a Harvard architecture with separate local instruction and data buses as well as a third bus for peripherals. The ARM Cortex-M3 CPU also includes an internal prefetch unit that supports speculative branching.

The peripheral complement of the LPC1769/68/67/66/65/64/63 includes up to 512 kB of flash memory, up to 64 kB of data memory, Ethernet MAC, USB Device/Host/OTG interface, 8-channel general purpose DMA controller, 4 UARTs, 2 CAN channels, 2 SSP controllers, SPI interface, 3 I^2C-bus interfaces, 2-input plus 2-output I^2S-bus interface, 8-channel 12-bit ADC, 10-bit DAC, motor control PWM, Quadrature Encoder interface, four general purpose timers, 6-output general purpose PWM, ultra-low power Real-Time Clock (RTC) with separate battery supply, and up to 70 general purpose I/O pins.

The LPC1769/68/67/66/65/64/63 are pin-compatible to the 100-pin LPC236x ARM7-based microcontroller series.

For additional documentation, see Section 19 "References".

2. Features and benefits

- ARM Cortex-M3 processor, running at frequencies of up to 100 MHz (LPC1768/67/66/65/64/63) or of up to 120 MHz (LPC1769). A Memory Protection Unit (MPU) supporting eight regions is included.
- ARM Cortex-M3 built-in Nested Vectored Interrupt Controller (NVIC).
- Up to 512 kB on-chip flash programming memory. Enhanced flash memory accelerator enables high-speed 120 MHz operation with zero wait states.
- In-System Programming (ISP) and In-Application Programming (IAP) via on-chip bootloader software.
- On-chip SRAM includes:
  - 32/16 kB of SRAM on the CPU with local code/data bus for high-performance CPU access.

- Two/one 16 kB SRAM blocks with separate access paths for higher throughput. These SRAM blocks may be used for Ethernet, USB, and DMA memory, as well as for general purpose CPU instruction and data storage.

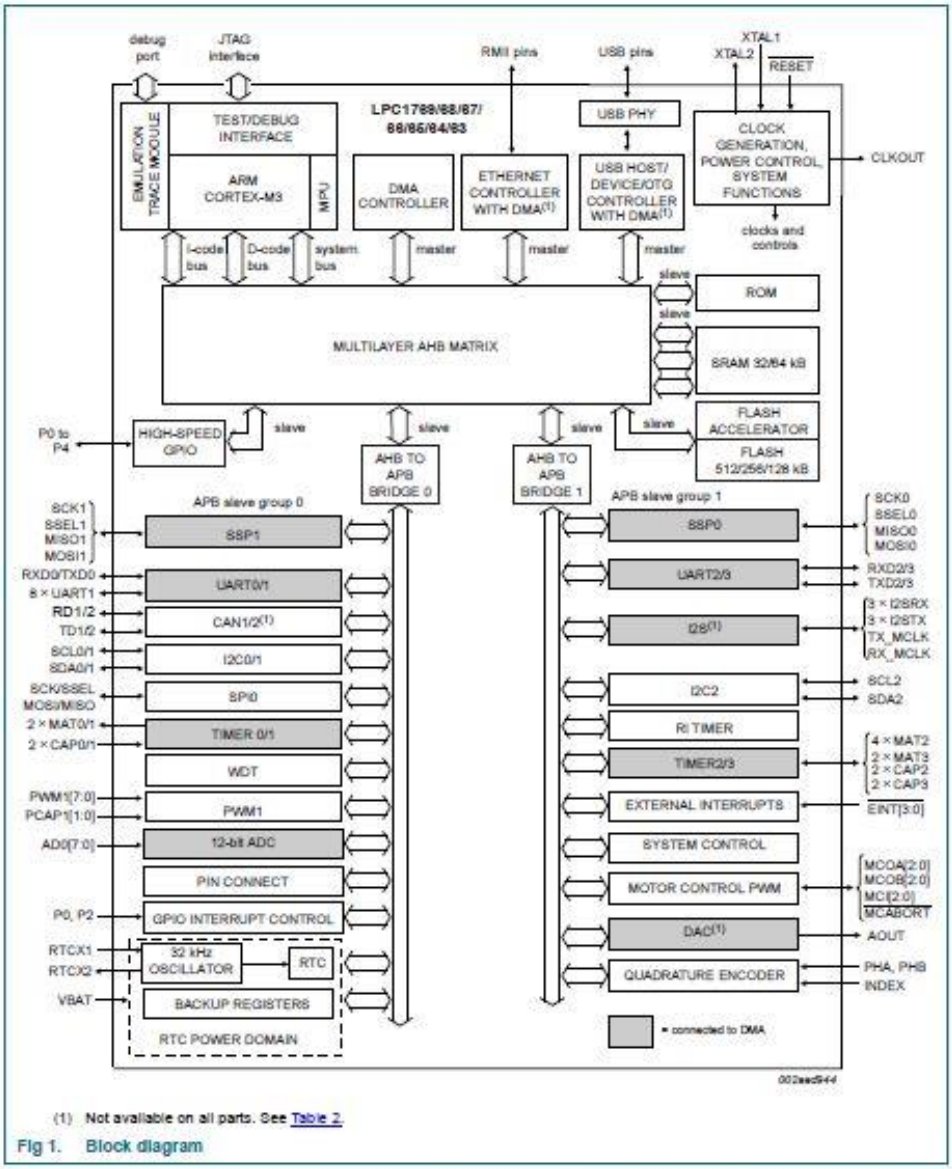
- Eight channel General Purpose DMA controller (GPDMA) on the AHB multilayer matrix that can be used with SSP, I²S-bus, UART, Analog-to-Digital and Digital-to-Analog converter peripherals, timer match signals, and for memory-to-memory transfers.
- Multilayer AHB matrix interconnect provides a separate bus for each AHB master. AHB masters include the CPU, General Purpose DMA controller, Ethernet MAC, and the USB interface. This interconnect provides communication with no arbitration delays.
- Split APB bus allows high throughput with few stalls between the CPU and DMA.
- Serial interfaces:
  - Ethernet MAC with RMII interface and dedicated DMA controller. (Not available on all parts, see Table 2.)
  - USB 2.0 full-speed device/Host/OTG controller with dedicated DMA controller and on-chip PHY for device, Host, and OTG functions. (Not available on all parts, see Table 2.)
  - Four UARTs with fractional baud rate generation, internal FIFO, and DMA support. One UART has modem control I/O and RS-485/EIA-485 support, and one UART has IrDA support.
  - CAN 2.0B controller with two channels. (Not available on all parts, see Table 2.)
  - SPI controller with synchronous, serial, full duplex communication and programmable data length.
  - Two SSP controllers with FIFO and multi-protocol capabilities. The SSP interfaces can be used with the GPDMA controller.
  - Three enhanced I²C bus interfaces, one with an open-drain output supporting full I²C specification and Fast mode plus with data rates of 1 Mbit/s, two with standard port pins. Enhancements include multiple address recognition and monitor mode.
  - I²S (Inter-IC Sound) interface for digital audio input or output, with fractional rate control. The I²S-bus interface can be used with the GPDMA. The I²S-bus interface supports 3-wire and 4-wire data transmit and receive as well as master clock input/output. (Not available on all parts, see Table 2.)
- Other peripherals:
  - 70 (100 pin package) General Purpose I/O (GPIO) pins with configurable pull-up/down resistors. All GPIOs support a new, configurable open-drain operating mode. The GPIO block is accessed through the AHB multilayer bus for fast access and located in memory such that it supports Cortex-M3 bit banding and use by the General Purpose DMA Controller.
  - 12-bit Analog-to-Digital Converter (ADC) with input multiplexing among eight pins, conversion rates up to 200 kHz, and multiple result registers. The 12-bit ADC can be used with the GPDMA controller.
  - 10-bit Digital-to-Analog Converter (DAC) with dedicated conversion timer and DMA support. (Not available on all parts, see Table 2)
  - Four general purpose timers/counters, with a total of eight capture inputs and ten compare outputs. Each timer block has an external count input. Specific timer events can be selected to generate DMA requests.
  - One motor control PWM with support for three-phase motor control.

- Quadrature encoder interface that can monitor one external quadrature encoder.
- One standard PWM/timer block with external count input.
- RTC with a separate power domain and dedicated RTC oscillator. The RTC block includes 20 bytes of battery-powered backup registers.
- WatchDog Timer (WDT). The WDT can be clocked from the internal RC oscillator, the RTC oscillator, or the APB clock.
- ARM Cortex-M3 system tick timer, including an external clock input option.
- Repetitive interrupt timer provides programmable and repeating timed interrupts.
- Each peripheral has its own clock divider for further power savings.

- Standard JTAG test/debug interface for compatibility with existing tools. Serial Wire Debug and Serial Wire Trace Port options.
- Emulation trace module enables non-intrusive, high-speed real-time tracing of instruction execution.
- Integrated PMU (Power Management Unit) automatically adjusts internal regulators to minimize power consumption during Sleep, Deep sleep, Power-down, and Deep power-down modes.
- Four reduced power modes: Sleep, Deep-sleep, Power-down, and Deep power-down.
- Single 3.3 V power supply (2.4 V to 3.6 V).
- Four external interrupt inputs configurable as edge/level sensitive. All pins on Port 0 and Port 2 can be used as edge sensitive interrupt sources.
- Non-maskable Interrupt (NMI) input.
- Clock output function that can reflect the main oscillator clock, IRC clock, RTC clock, CPU clock, and the USB clock.
- The Wake-up Interrupt Controller (WIC) allows the CPU to automatically wake up from any priority interrupt that can occur while the clocks are stopped in deep sleep, Power-down, and Deep power-down modes.
- Processor wake-up from Power-down mode via any interrupt able to operate during Power-down mode (includes external interrupts, RTC interrupt, USB activity, Ethernet wake-up interrupt, CAN bus activity, Port 0/2 pin interrupt, and NMI).
- Brownout detect with separate threshold for interrupt and forced reset.
- Power-On Reset (POR).
- Crystal oscillator with an operating range of 1 MHz to 25 MHz.
- 4 MHz internal RC oscillator trimmed to 1 % accuracy that can optionally be used as a system clock.
- PLL allows CPU operation up to the maximum CPU rate without the need for a high-frequency crystal. May be run from the main oscillator, the internal RC oscillator, or the RTC oscillator.
- USB PLL for added flexibility.
- Code Read Protection (CRP) with different security levels.
- Unique device serial number for identification purposes.
- Available as LQFP100 (14 mm × 14 mm × 1.4 mm), TFBGA100[1] (9 mm × 9 mm × 0.7 mm), and WLCSP100 (5.074 × 5.074 × 0.6 mm) package.

1. LPC1768/65 only.

6. Block diagram

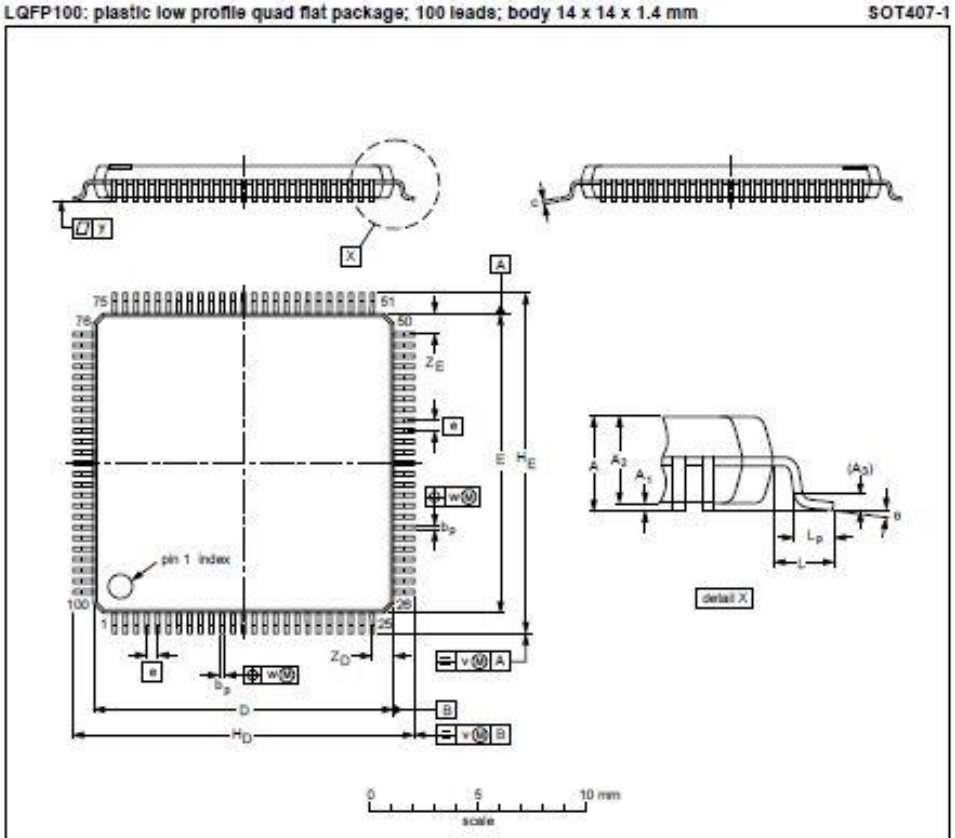


(1) Not available on all parts. See Table 2.

Fig 1. Block diagram

16. Package outline

LQFP100: plastic low profile quad flat package; 100 leads; body 14 x 14 x 1.4 mm SOT407-1



DIMENSIONS (mm are the original dimensions)

UNIT	A max.	A_1	A_2	A_3	b_p	c	$D^{(1)}$	$E^{(1)}$	e	H_D	H_E	L	L_p	v	w	y	$Z_D^{(1)}$	$Z_E^{(1)}$	θ
mm	1.6	0.15 0.05	1.45 1.35	0.25	0.27 0.17	0.20 0.09	14.1 13.9	14.1 13.9	0.5	16.25 15.75	16.25 15.75	1	0.75 0.45	0.2	0.08	0.08	1.15 0.85	1.15 0.85	7° 0°

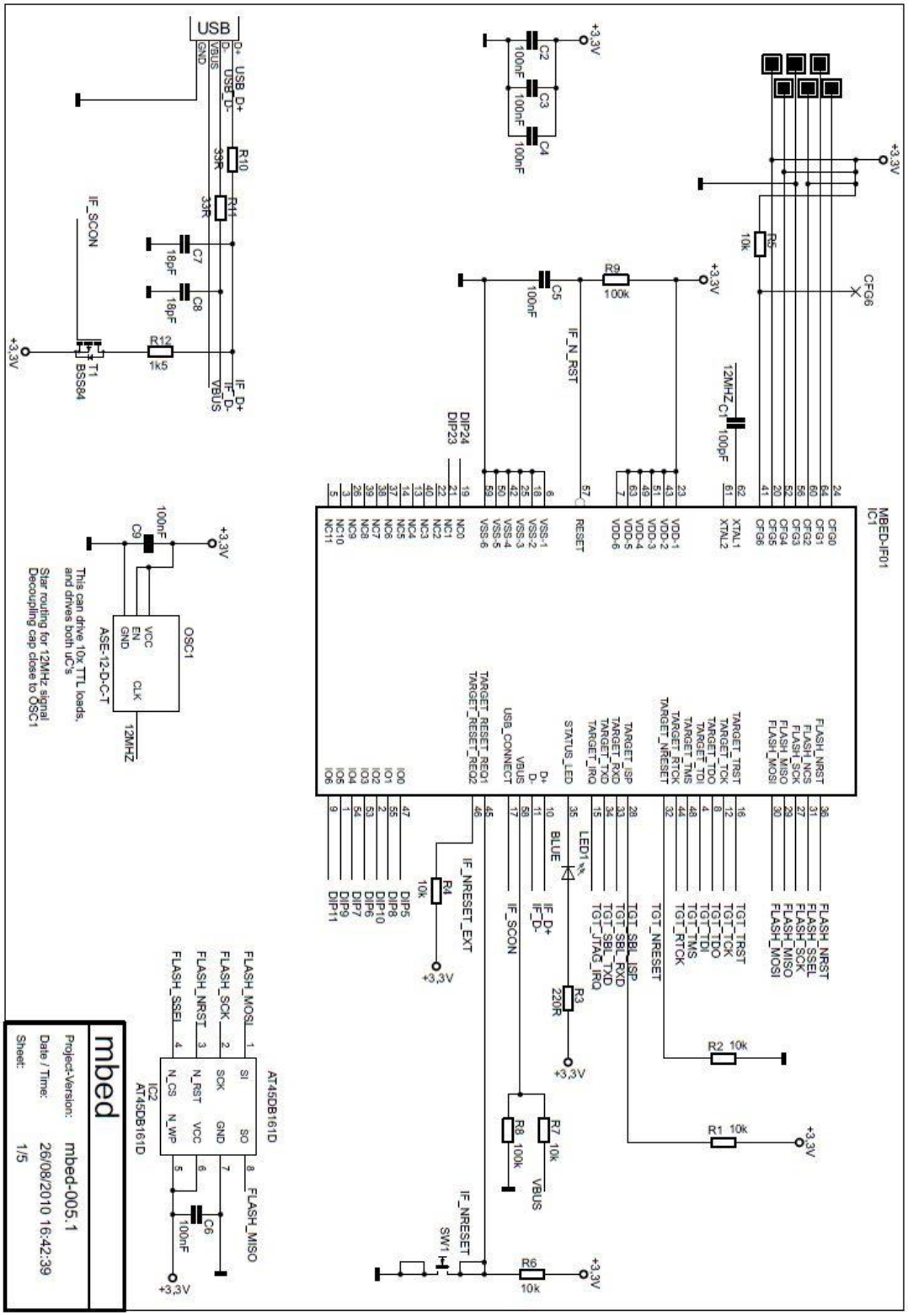
Note

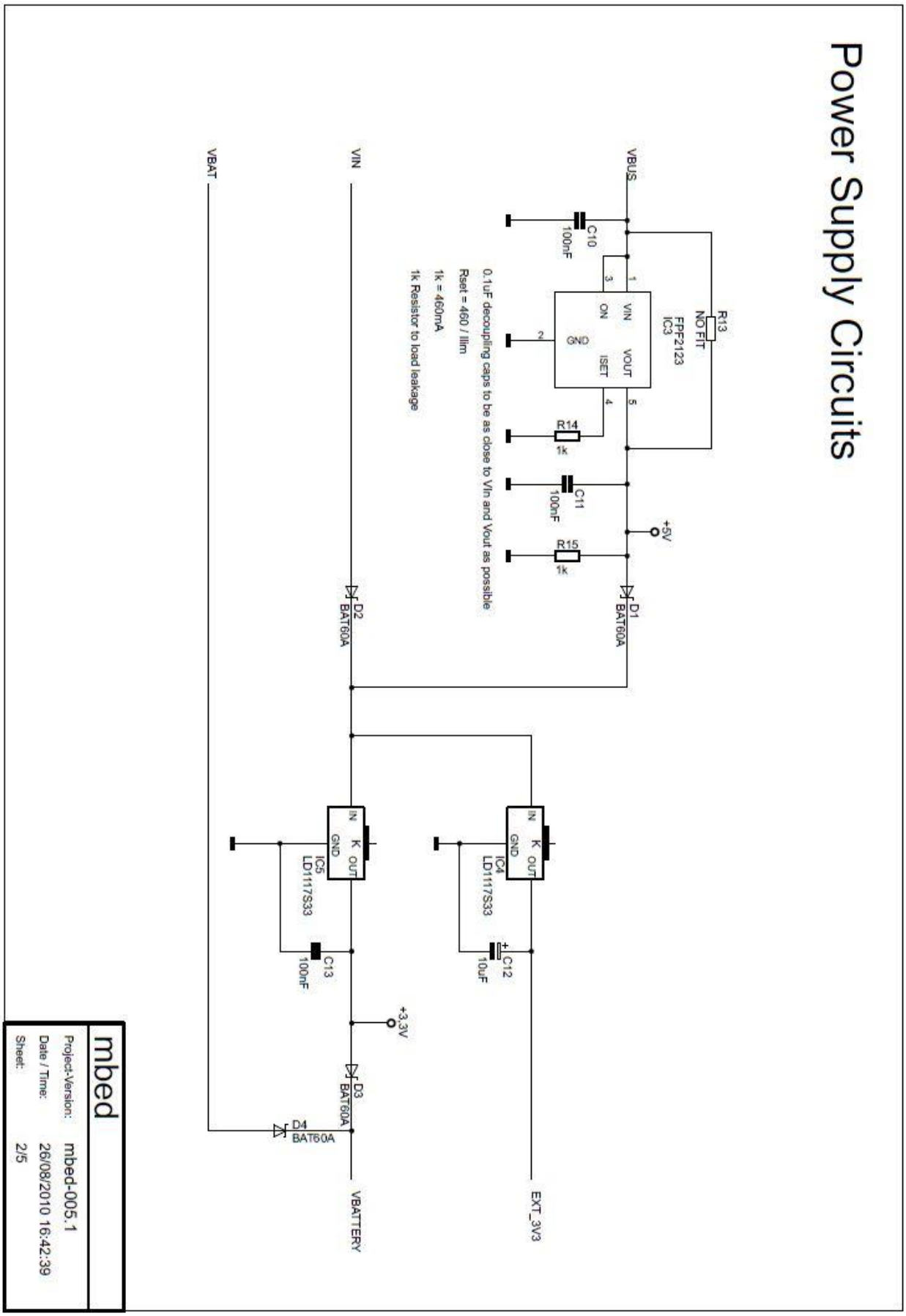
1. Plastic or metal protrusions of 0.25 mm maximum per side are not included.

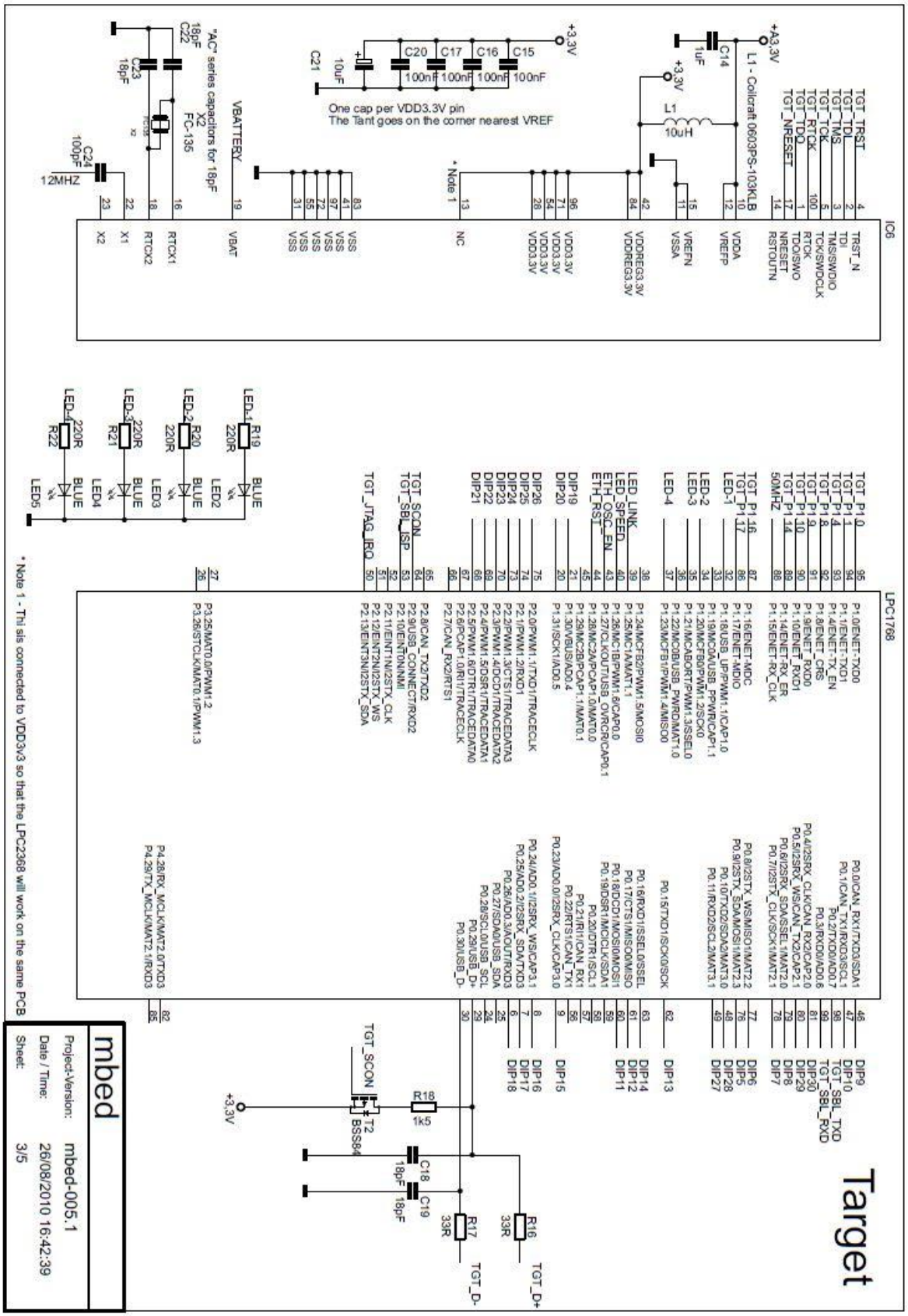
OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT407-1	136E20	MS-026				~~00-01-01~~ 03-02-20

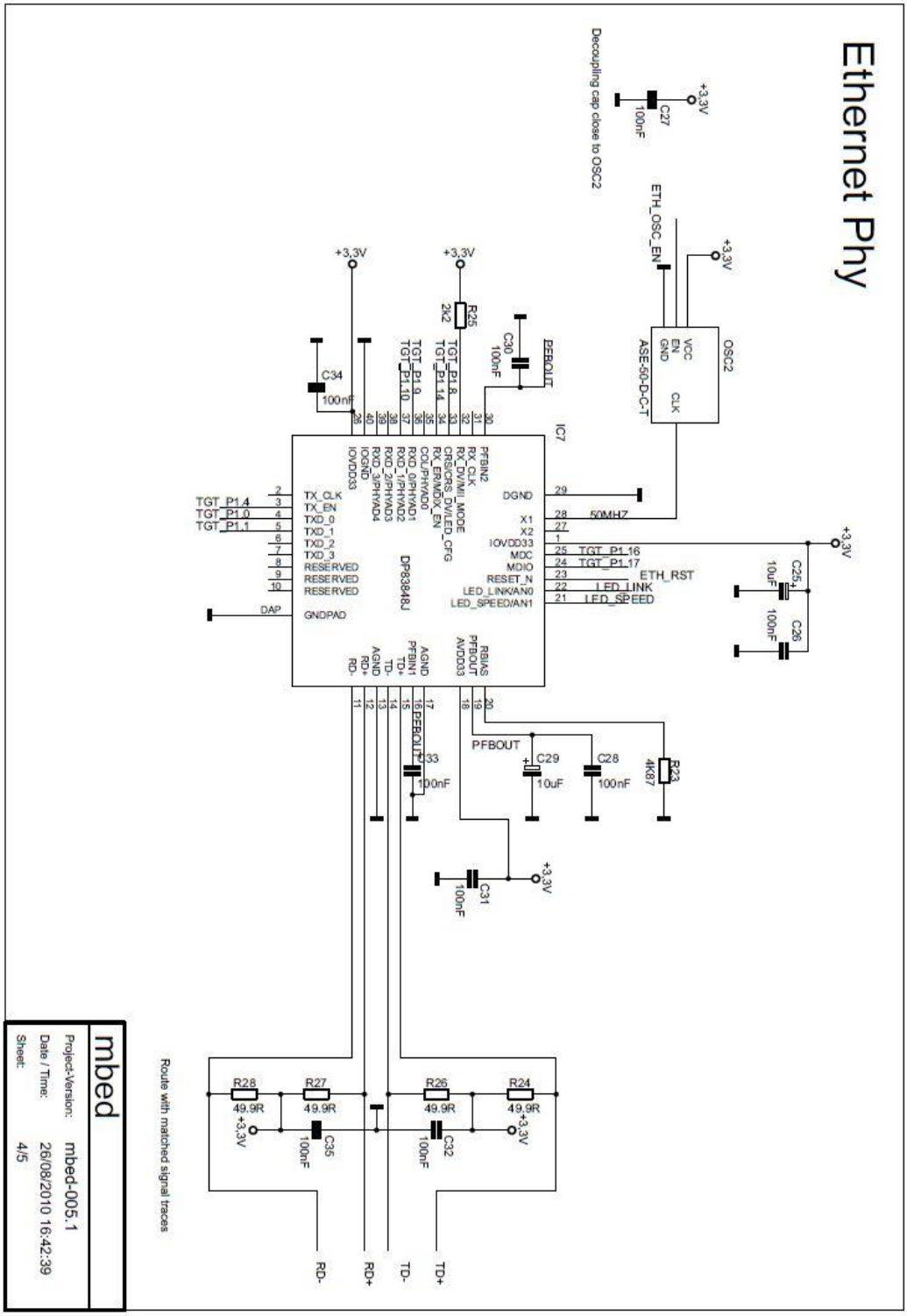
Fig 40. Package outline SOT407-1 (LQFP100)

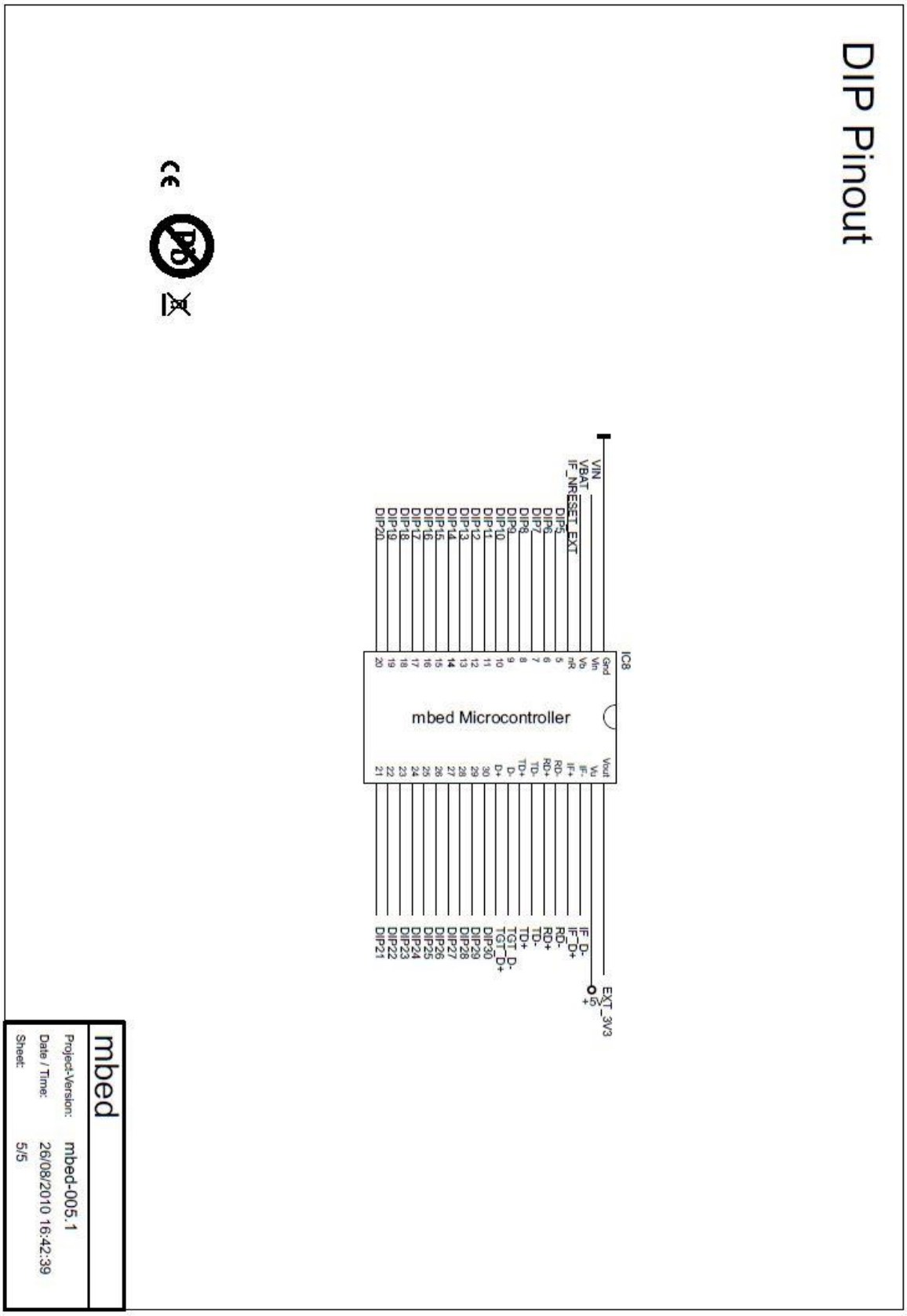
6.5 – Anexo V. Esquemático MBED LCP 1768




Power Supply Circuits
VBUS
VIN
VBAT
C10
100nF
R13
NO FIT
FPF2123
IC3
VIN
ON
GND
VOUT
ISET
R14
1k
C11
100nF
R15
1k
+5V
D1
BAT60A
D2
BAT60A
0.1uF decoupling caps to be as close to Vin and Vout as possible
Rset = 460 / Ilim
1k = 460mA
1k Resistor to load leakage
IC4
LD1117S33
IC5
LD1117S33
C12
10uF
C13
100nF
+3.3V
D3
BAT60A
D4
BAT60A
EXT_3V3
VBATTERY
mbed
Project-Version: mbed-005.1
Date / Time: 26/08/2010 16:42:39
Sheet: 2/5


mbed
Project-Version: mbed-005.1
Date / Time: 26/08/2010 16:42:39
Sheet: 3/5
Target
LPC1768
IC6
* Note 1 - Thi sis connected to VDD3v3 so that the LPC2368 will work on the same PCB
One cap per VDD3.3V pin
The Tant goes on the corner nearest VREF
"AC" series capacitors for 18pF
X2
FC-135
12MHZ
L1 - Coilcraft 0603PS-103KLB
L1
10uH
C14
1uF
C15
C16
C17
C20
100nF
C21
10uF
C22
18pF
C23
18pF
C24
100pF
VBATTERY
+3.3V
+A3.3V
TGT_TRST
TGT_TDI
TGT_TMS
TGT_TCK
TGT_RTCK
TGT_TDO
TGT_NRESET
TRST_N
TDI
TMS/SWDIO
TCK/SWDCLK
RTCK
TDO/SWO
NRESET
RSTOUTN
VDDA
VREFP
VREFN
VSSA
VDDREG3.3V
VDD3.3V
VSS
NC
RTCX1
RTCX2
X1
X2
VBAT
LED-1
LED-2
LED-3
LED-4
50MHZ
TGT_P1.16
TGT_P1.17
DIP19
DIP20
DIP21
DIP22
DIP23
DIP24
DIP25
DIP26
LED_LINK
LED_SPEED
ETH_OSC_EN
ETH_RST
TGT_SCON
TGT_SBL_ISP
TGT_JTAG_IRQ
LED1
LED2
LED3
LED4
LED5
R19
R20
R21
R22
220R
BLUE
P0.0/CAN_RX1/TXD3/SDA1
P0.1/CAN_TX1/RXD3/SCL1
P0.27/TXD0/AD0.7
P0.3/RXD0/AD0.6
P0.4/I2SRX_CLK/CAP2.0
P0.5/I2SRX_WS/CAP2.1
P0.6/I2SRX_SDA/SSEL1/MAT2.0
P0.7/I2STX_CLK/SCK1/MAT2.1
P0.8/I2STX_WS/MISO1/MAT2.2
P0.9/I2STX_SDA/MOSI1/MAT2.3
P0.10/TXD2/SDA2/MAT3.0
P0.11/RXD2/SCL2/MAT3.1
P0.15/TXD1/SCK0/SCK
P0.16/RXD1/SSEL0/SSEL
P0.17/CTS1/MISO0/MISO
P0.18/DCD1/MOSI0/MOSI
P0.19/DSR1/MCICLK/SDA
P0.20/DTR1/SCL1
P0.21/RI1/CAN_RX1
P0.22/RTS1/CAN_TX1
P0.23/AD0.0/I2SRX_CLK/CAP3.0
P0.24/AD0.1/I2SRX_WS/CAP3.1
P0.25/AD0.2/I2SRX_SDA/TXD3
P0.26/AD0.3/AOUT/RXD3
P0.27/SDA0/USB_SDA
P0.28/SCL0/USB_SCL
P0.29/USB_D+
P0.30/USB_D-
P4.28/RX_MCLK/MAT2.0/TXD3
P4.29/TX_MCLK/MAT2.1/RXD3
P3.25/MAT0.0/PWM1.2
P3.26/STCLK/MAT0.1/PWM1.3
P2.8/CAN_TX2/TXD2
P2.9/USB_CONNECT/RXD2
P2.10/EINT0/NMI
P2.11/EINT1/I2STX_CLK
P2.12/EINT2/I2STX_WS
P2.13/EINT3/I2STX_SDA
P1.0/ENET-TXD0
P1.1/ENET-TXD1
P1.4/ENET-TX_EN
P1.8/ENET_CRS
P1.9/ENET_RXD0
P1.10/ENET_RXD1
P1.14/ENET-RX_ER
P1.15/ENET-RX_CLK
P1.16/ENET-MDC
P1.17/ENET-MDIO
P1.18/USB_UP_PWM1.1/CAP1.0
P1.19/MCOA/USB_PPWR/CAP1.1
P1.20/MCFB0/PWM1.2/SCK0
P1.21/MCABORT/PWM1.3/SSEL0
P1.22/MCOB/USB_PWRD/MAT1.0
P1.23/MCFB1/PWM1.4/MISO0
P1.24/MCFB2/PWM1.5/MOSI0
P1.25/MC1A/MAT1.1
P1.26/MC1B/PWM1.6/CAP0.0
P1.27/CLKOUT/USB_OVRCR/CAP0.1
P1.28/MC2A/PCAP1.0/MAT0.0
P1.29/MC2B/PCAP1.1/MAT0.1
P1.30/VBUS/AD0.4
P1.31/SCK1/AD0.5
P2.0/PWM1.1/TXD1/TRACECLK
P2.1/PWM1.2/RXD1
P2.2/PWM1.3/CTS1/TRACEDATA3
P2.3/PWM1.4/DCD1/TRACEDATA2
P2.4/PWM1.5/DSR1/TRACEDATA1
P2.5/PWM1.6/DTR1/TRACEDATA0
P2.6/PCAP1.0/RI1/TRACECLK
P2.7/CAN_RX2/RTS1
DIP9
DIP10
TGT_SBL_TXD
TGT_SBL_RXD
DIP30
DIP29
DIP8
DIP7
DIP6
DIP5
DIP28
DIP27
DIP13
DIP14
DIP12
DIP11
DIP15
DIP16
DIP17
DIP18
R16
R17
33R
TGT_D+
TGT_D-
R18
1k2
T2
BSS84
C18
C19
18pF


Ethernet Phy
Decoupling cap close to OSC2
C27
100nF
+3,3V
ETH_OSC_EN
OSC2
ASE-50-D-C-T
VCC
EN
GND
CLK
R25
2k2
C30
100nF
PFBOUT
C34
100nF
IC7
DP83848J
TX_CLK
TX_EN
TXD_0
TXD_1
TXD_2
TXD_3
RESERVED
RESERVED
RESERVED
DAP
GNDPAD
DGND
X1
X2
IOVDD33
MDC
MDIO
RESET_N
LED_LINK/AN0
LED_SPEED/AN1
50MHZ
TGT_P1.16
TGT_P1.17
ETH_RST
LED_LINK
LED_SPEED
TGT_P1.4
TGT_P1.0
TGT_P1.1
C25
10uF
C26
100nF
C29
10uF
C28
100nF
R23
4K87
C33
100nF
C31
100nF
R28
R27
R26
R24
49.9R
C35
C32
100nF
RD-
RD+
TD-
TD+
Route with matched signal traces
mbed
Project-Version: mbed-005.1
Date / Time: 26/08/2010 16:42:39
Sheet: 4/5


DIP Pinout
mbed Microcontroller
IC8
VIN
VBAT
IF_NRESET_EXT
DIP5
DIP6
DIP7
DIP8
DIP9
DIP10
DIP11
DIP12
DIP13
DIP14
DIP15
DIP16
DIP17
DIP18
DIP19
DIP20
Gnd
Vin
Vb
nR
Vu
IF-
IF+
RD-
RD+
TD-
TD+
D-
D+
TGT_D-
TGT_D+
Vout
IF_D-
IF_D+
RD-
RD+
TD-
TD+
DIP30
DIP29
DIP28
DIP27
DIP26
DIP25
DIP24
DIP23
DIP22
DIP21
EXT_3V3
+5
mbed
Project-Version: mbed-005.1
Date / Time: 26/08/2010 16:42:39
Sheet: 5/5